

西安交通大学

博士学位论文

网络数据平面可编程硬件的研究

学位申请人：乔思祎

指导教师：邹建华 教授

合作导师：胡成臣 教授 管晓宏 教授

学科名称：控制科学与工程

2020 年 10 月

Research on Programmable Hardware for Network Data Plane

A dissertation submitted to
Xi'an Jiaotong University
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

By

Siyi Qiao

Supervisor: Prof. Jianhua Zou

Associate Supervisor: Prof. Chengchen Hu and Xiaohong Guan

Automation Science and Engineering

September 2020

博士学位论文答辩委员会

网络数据平面可编程硬件的研究

答辩人：乔思祎

答辩委员会委员：

西安交通大学 xx 教授：_____ (主席)

西安交通大学 xx 教授：_____

西安交通大学 xx 教授：_____

西安交通大学 xx 教授：_____

西安交通大学 xx 教授：_____

答辩时间：xxxx 年 xx 月 xx 日

答辩地点：地点 xx

摘要

软件定义网络概念的提出，推动网络向高性能和高可编程的方向发展。软件定义网络利用流表建立起“匹配 + 执行”的编程抽象方法，解耦网络数据平面与控制平面；数据包协议无关处理器通过扩展匹配域定义的方式，进一步优化了网络可编程抽象能力。可编程网络架构的控制平面与数据平面分离，数据平面转发设备中的流表是表达控制算法灵活性的关键部件。但高性能硬件流表容量较低，易导致节点流表溢出从而影响转发性能；数据包协议无关的处理技术扩展了数据包头的匹配灵活度，但无法满足数据平面内其他可编程计算需求。因此目前网络数据平面设备面临着吞吐性能和灵活性两方面的制约。

本文主要研究在网络数据平面内如何利用可编程硬件来更灵活、更高效地支持多种类别的可编程抽象方法。本文首先以解决流表资源瓶颈问题为基础，接着利用可编程硬件分别在网络中间节点、主机侧网络中构建高性能与高可编程性协同发展的方法，最终形成一套完整的端到端网络系统。具体贡献如下：

1. 提出一种全局场景下的流表资源可扩展方法（Flow Table Sharing, FTS）。为解决单个转发节点流表容量受限以及流表溢出后数据包处理性能严重下降的问题，本文提出流表共享机制 FTS。FTS 通过与邻居节点建立流表共享，提升整体流表资源的动态利用率，同时为由于没有足够空间而无法建立流表项的流量，设置一种基于离线转发策略的组表转发方式。在流表溢出的情况下，该方案和传统 OpenFlow 协议相比，使 OpenFlow 交换机转发 RTT 延迟和安全通道控制报文风暴数量的优化均达到至少 2 个数量级。
2. 支持自适应计算的硬件交换系统架构（Adaptable Switch, AS）。针对高性能网络转发设备可编程灵活性差的问题，本文提出利用可编程硬件加速网络中间节点的普适性计算的方法 AS。利用高性能的转发芯片与可编程硬件有机结合形成异构体，同时获得性能提升以及更高的可编程灵活性。并且还提出了一套部署在硬件上的高资源利用率的并行流水线和流表分配优化算法，使自适应交换系统在满足全可编程灵活性的条件下，与基于 FPGA 的数据平面相比，将数据包处理性能提升 120 倍以上。
3. 端侧网络在网络测量领域的可编程抽象方法。网络测量是众多网络功能的基础，针对目前测量工具性能低下以及缺乏统一的编程抽象，为了提高系统使用灵活性以及可用性，本文针对大量场景提出了一套基于测量技术的网卡硬件流水线系统。通过不同的软件调用，令系统硬件适用于高性能的网络安全、访问控制、流量控制、拥塞探测等多种场景。考虑到高性能硬件资源有限的问题，论文将可编程测量系统抽象为基础的包个数与数据量的压缩统计并提供不同的统计触发方法。在节约 38% 的硬件存储空间情况下，相较于软件的处理方式系统吞吐率提升 8 倍、系统的处理能耗节约 90%。

关键词：软件定义网络；网络数据平面；可编程硬件；现场可编程门阵列；流表

论文类型: 应用基础

ABSTRACT

The concept of software-defined networking has promoted the development of high-performance and high-programmability networks. The software-defined network uses flow tables to establish an abstract programming method of "matching + execution" to decouple the network data plane and control plane; The packet protocol-independent processor further optimizes the network programmable abstraction ability by extending the matching domain definition. The control plane of the programmable network architecture is separated from the data plane, and the flow table in the data plane forwarding device is a key component that expresses the flexibility of the control algorithm. However, the high-performance hardware flow table has a low capacity, which can easily cause the node flow table to overflow and affect the forwarding performance; The data packet protocol-independent processing technology expands the matching flexibility of the data packet header, but cannot meet other programmable computing requirements in the data plane. Therefore, the current network data plane equipment is facing the constraints of throughput performance and flexibility.

This article mainly studies how to use programmable hardware in the network data plane to support multiple types of programmable abstraction methods more flexibly and efficiently. This article is based on solving the bottleneck problem of flow table resources, and uses programmable hardware to build a coordinated development of high performance and high programmability methods in the network intermediate nodes and the host side network, and finally form a complete end-to-end network system. The specific contributions are as follows:

1. Propose an extensible method for flow table resources (Flow Table Sharing, FTS) in a global scenario. In order to solve the problem that the capacity of the flow table of a single forwarding node is limited and the performance of data packet processing after the flow table overflows, the flow table sharing mechanism FTS is proposed in this paper. FTS establishes flow table sharing with neighbor nodes to improve the dynamic utilization of the overall flow table resources. At the same time, it sets a group table forwarding method based on an offline forwarding strategy for the traffic that cannot establish flow entries due to insufficient space. In the case of flow table overflow, compared with the traditional OpenFlow protocol, this solution enables the optimization of the OpenFlow switch forwarding RTT delay and the number of secure channel control packet storms to reach at least 2 orders of magnitude.
2. Supporting adaptive computing hardware switching system architecture (Adaptable Switch, AS). Aiming at the problem of poor programmable flexibility of high-performance network

forwarding equipment, this paper proposes a method AS, which uses programmable hardware to accelerate the universal computing of network intermediate nodes. Use high-performance forwarding chips and programmable hardware to organically combine to form isomers, while gaining performance improvements and higher programmable flexibility. It also proposes a set of parallel pipelines and flow table allocation optimization algorithms with high resource utilization deployed on hardware, so that the adaptive switching system meets the conditions of full programmable flexibility, compared with FPGA-based data planes. Improve data packet processing performance by more than 120 times.

3. The programmable abstract method of end-side network in the field of network measurement. Network measurement is the basis of many network functions. In view of the low performance of current measurement tools and the lack of unified programming abstraction, in order to improve the flexibility and usability of the system, this paper proposes a network card hardware pipeline system based on measurement technology for a large number of scenarios. Through different software calls, the system hardware is suitable for high-performance network security, access control, flow control, congestion detection and other scenarios. Considering the limited resources of high-performance hardware, The thesis abstracts the programmable measurement system based on the compression statistics of the number of packets and data volume and provides different statistical trigger methods.

In the case of saving 38% of the hardware storage space, the system throughput rate is increased by 8 times compared with the software processing method, and the processing energy consumption of the system is reduced by 90%.

KEY WORDS: Software Defined network, Data Plane, Hardware-based programmability, FPGA, Flow Table

TYPE OF DISSERTATION: Application Fundamentals

目 录

摘要.....	I
ABSTRACT	III
1 绪论.....	1
1.1 研究背景与研究意义	1
1.1.1 网络数据平面概述.....	1
1.1.2 网络可编程技术.....	2
1.1.3 可编程网络的关键问题.....	5
1.2 网络数据平面的研究现状	6
1.2.1 网络数据平面的运算资源问题.....	6
1.2.2 网络数据平面的编程能力问题.....	9
1.2.3 网络数据平面的编程抽象问题.....	11
1.3 论文研究内容与创新点.....	12
1.3.1 研究挑战	12
1.3.2 本文研究内容与创新点.....	13
1.4 论文组织结构	16
2 SDN 网络交换机硬件流表可扩展性研究	18
2.1 引言	18
2.2 问题背景概述	18
2.3 基于流量特征的问题分析	20
2.3.1 流量细粒度趋势	21
2.3.2 问题分析	21
2.4 流表共享机制	24
2.4.1 允许流表溢出时交换机转发新流.....	24
2.4.2 减小重复控制消息的数量.....	25
2.5 基于 OpenFlow 交换机的随机路由策略	25
2.5.1 随机路由离线策略设计	26
2.5.2 随机路由策略组表方式实现	26
2.6 系统评估	29
2.6.1 性能测试：转发时延与控制消息数量	30
2.6.2 代价评估：额外消耗全局资源与流路径.....	31

2.7 本章小结	34
3 研究可编程硬件设备加速网络交换层方法	35
3.1 引言	35
3.2 问题背景概述	35
3.3 系统架构介绍	37
3.3.1 自适应交换架构设计	37
3.3.2 用户定义功能的开发流程	39
3.3.3 高层次语言映射样例	41
3.4 自适应交换架构的硬件设计	42
3.4.1 数据平面内协议设计	42
3.4.2 SS 端固定功能	43
3.4.3 PL 设计	44
3.5 流表、流量分配问题	49
3.5.1 流量均衡的问题分析	49
3.5.2 NP 难问题证明	50
3.6 资源优化问题的启发式算法设计	53
3.7 系统开发以及测试	56
3.7.1 拥塞控制	57
3.7.2 网络测量	57
3.7.3 有状态防火墙	57
3.7.4 资源消耗	58
3.7.5 控制平面计算量	59
3.8 本章小结	60
4 端侧网络在网络测量领域的可编程抽象方法	62
4.1 引言	62
4.2 问题背景	63
4.3 系统架构分析	64
4.3.1 软件与专用硬件处理器的特性分析	64
4.3.2 测量系统软硬件架构	65
4.3.3 计算流水线的优化设计方法	66
4.4 网络流量高性能数据通路	67
4.4.1 概述	67

4.4.2 问题分析	68
4.4.3 设计	69
4.4.4 电路实现以及协议	70
4.4.5 优化	72
4.5 基于硬件的高性能压缩统计方法	73
4.5.1 网络统计方法概述	73
4.5.2 问题分析	74
4.5.3 基于硬件设计压缩效果与问题	74
4.5.4 优化	77
4.6 软硬一体化的系统实验平台	86
4.6.1 软件	86
4.6.2 硬件	87
4.7 系统评价	88
4.7.1 网络流量捕获与回放系统评估	88
4.7.2 网络测量实时压缩系统评估	90
4.8 本章小结	93
5 结论与展望	94
5.1 工作总结与主要成果	94
5.2 研究内容展望	95
5.2.1 数据平面可编程网络	95
5.2.2 网络资源与计算能力	95
致 谢	97
参考文献	98
攻读学位期间取得的研究成果	107
声 明	

CONTENTS

ABSTRACT (Chinese)	I
ABSTRACT (English)	III
1 Introduction	1
1.1 Research background and research significance	1
1.1.1 Network data plane overview	1
1.1.2 Network programmable technology	2
1.1.3 Key Issues of Programmable Networks	5
1.2 Research status of network data plane	6
1.2.1 Computing resource problem of network data plane	6
1.2.2 The programming ability of the network data plane	9
1.2.3 Programming abstraction of network data plane	11
1.3 Thesis research content and innovation	12
1.3.1 Research challenge	12
1.3.2 The research content and innovation of this article	13
1.4 Organization of the Thesis	16
2 Research on Scalability of SDN Hardware Flow Table	18
2.1 Introduction to Chapter	18
2.2 Background	18
2.3 Problem Analysis Based on Traffic Characteristics	20
2.3.1 Flow Fine-grained Trend	21
2.3.2 Problem Analysis	21
2.4 Flow table Sharing Mechanism	24
2.4.1 Allow the Switch to Forward New Flows When the Flow Table Overflows ..	24
2.4.2 Reducing the Number of Repeated Control Messages	25
2.5 Random Routing Strategy Based on OpenFlow Switch	25
2.5.1 Random Routing Offline Strategy design	26
2.5.2 Random routing strategy group table implementation	26
2.6 Evaluation	29
2.6.1 Performance Evaluation	30
2.6.2 Overhead Evaluation	31

CONTENTS

2.7 Conclusion	34
3 Research on Programmable Devices to Accelerate Hardware-based Core layer	35
3.1 Introduction to Chapter	35
3.2 Background	35
3.3 System Architecture Introduction	37
3.3.1 Architecture Design	37
3.3.2 Develop Flow.....	39
3.3.3 Examples of Mapping High Level Languages	41
3.4 Hardware Design	42
3.4.1 Protocol Design	42
3.4.2 Fixed Functions in SS	43
3.4.3 PL Design	44
3.5 Allocation Problem of Table Entries and Flow.....	49
3.5.1 Problem Analysis	49
3.5.2 NP hard proof	50
3.6 Algorithm Design	53
3.7 System Performance and Evaluation	56
3.7.1 Congestion Control	57
3.7.2 Network Measurement	57
3.7.3 A Stateful Firewall.....	57
3.7.4 Resource Consumption.....	58
3.7.5 The Calculation in Control Plane	59
3.8 Conclusion	60
4 Research on Programmable Equipment Accelerating Host Side Network.....	62
4.1 Introduction to Chapter	62
4.2 Background	63
4.3 System Architecture.....	64
4.3.1 Analysis of Software Hardware Offloading	64
4.3.2 Introduction	65
4.3.3 Hardware Abstraction Method of Software Algorithm	66
4.4 Network Traffic Capture and Replay	67
4.4.1 Introduction.....	67
4.4.2 Problem Analysis	68

4.4.3 System Design.....	69
4.4.4 Circuit Implementation and Protocol.....	70
4.4.5 Optimization.....	72
4.5 Statistics—Real-time Compression for Network Measurement	73
4.5.1 Introduction.....	73
4.5.2 Problem Analysis	74
4.5.3 Compression Effects and Problems Based on Hardware Design.....	74
4.5.4 Optimization.....	77
4.6 Software and Hardware Co-design Platform	86
4.6.1 Software	86
4.6.2 Hardware	87
4.7 Evaluation	88
4.7.1 Evaluation of Traffic Capture and Replay	88
4.7.2 Evaluation of Network Measurement	90
4.8 Conclusion	93
5 Conclusions and Future Works	94
5.1 Thesis Conclusion	94
5.2 Future Works	95
5.2.1 Data of Plane Programmable Network.....	95
5.2.2 Network Resources and In-Network Computing	95
Acknowledgements.....	97
References	98
Achievements	107
Declarations	

1 绪论

1.1 研究背景与研究意义

1.1.1 网络数据平面概述

网络正以前所未有的速度越来越紧密地参与到民生社会中，为满足国家民生需求、新基建拉动内需和产业升级起到了至关重要的作用。从“百度一下”到网红全民直播带货，从实现“三网通”到发展“新基建”的国家战略，小到优化社会资源效率的办公数字化，大到勾勒出智能交通、智慧城市和万物互联的5G海洋，无一不是构建在网络基础设施的快速发展之上。思科公司预计，到2023年全球家用互联网总带宽将达到5.85Ebps^①（是现在的3.27倍），移动互联网用户预计达到57亿，其总流量可达11.3Ebps（将达到目前的5倍），其中5G流量将占据移动互联网总带宽的76.5%（0.6%，2019年）^[1-2]。由于深度学习、AI、大数据、云计算、物联网的快速发展^[3-4]，这些新技术将促使新零售、新金融、新医疗、新教育、新制造、云视频和云游戏等行业“云化”，海量的数据会在数据中心内部服务器间网络以及外部网关中传递，这些关键应用将会改变数据中心算力构成和数据中心内部网络结构特性。

随着云服务概念和大规模机器学习的落地，以云计算为代表的数据中心网络规模成指数增长。同时，众核CPU架构快速发展，服务器内虚拟机布置资源大幅扩张，促使主机出口吞吐量从40GbE向100GbE甚至400GbE演进。此外，复杂的网络安全规则、流量监控等功能模块进一步吞噬了大量宝贵的CPU计算资源。为应对以上纷繁复杂的网络功能需求，近年来网络数据平面架构的发展尤为迅速^[5-8]。传统网络以TCP/IP协议为核心，主要目标为增强网络扩容能力，主要展现了网络强大的敏捷性与自治能力。但当网络规模进一步增加，网络中容易出现广播风暴、链路收敛慢等问题。大型的核心网交换设备往往需要具有能够应对各种复杂应用场景的能力，这使得网络设备结构越来越复杂、设备内的功能众多而应用场景单一，最终导致了设备成本高昂、设备内程序运行资源浪费巨大的问题。由于研究的创新和技术的推进，设备厂商不断开发出具备各种高级功能的交换芯片，硬件功能强大的同时，复杂的网络管理也遇到新的挑战。

“可编程网络”概念为网络架构带来了统一透明的控制接口，以及方便敏捷的网络管理能力。作为可编程网络的先驱，软件定义网络（Software Defined Network，SDN^[9]）概念首次在2008年被斯坦福大学提出，其将大学校园网内交换设备的控制平面抽离出来，以增强网络管理配置灵活度以及网络的快速排错能力。到目前为止，世界各国都投入大量资源进行软件定义网络的技术研究与开发。2010年，阿里巴巴云网络飞天系统在大规模和低成本的要求下，利用SDN技术提高云网络的可靠性和使用弹性。2013年，Google公司的全球数据中心骨干网使用SDN技术将网络链路的利用率从75%提升至99%^[10]。2015年至今，阿尔卡特朗讯在欧洲建立了Nuage^[11]项目，主要旨在运营商、

^① 1 Ebps=10⁶Tbps=10¹⁸bps

银行以及数据中心内部署 SDN 方案以增强网络基础设施的性能以及开发便捷性。2016 年，中国盛科网络技术公司领导研发的“支持软件定义网络千兆以太网交换芯片”通过了由中科院院士、“核高基”专家、电信专家在内的鉴定委员会评审，旨在制造中国具有全球领先技术的可编程网络解决方案^[12]。2019 年，中国联通通信公司完成国内首个基于 SDN 的互联网智能差异化服务商用部署，以应对 4K/8K 视频、VR、云应用等业务等对网络提出的多元化需求^[13]。对于网络节点数量大的复杂通信网络而言，集中控制可以有效地解决分布式信息同步缓慢，大规模网络问题排查困难等问题^[14-15]。可编程网络的概念从学术到商用落地，均展现了它强大的技术优势以及产业界广泛的认同。

1.1.2 网络可编程技术

如图 1-1 左部所示，按照网络转发设备的功能划分，可将网络设备从逻辑上分为“控制平面和数据平面”。传统网络架构中，一般将数据平面与控制平面集中在一起组成完整的设备。根据组网需求的不同，开发人员将同类型的网络设备互相连接在一起。每个设备内的控制平面会按照事先规定的协议算法与邻居节点或远端节点进行通信，同步完消息之后，每个设备节点都会拥有全局的网络拓扑数据，并在数据平面内安装根据路径选择好的转发策略。然而，当网络节点数量增加，网络拓扑连接变得复杂之后，传统的网络架构会面临网络拓扑信息共享速度缓慢等问题。同时，由于控制平面内均构建了自主运行的控制算法，网络管理员无法及时获取设备中实时的运行状态，对于当网络中出现任何问题时，开发人员很难精准定位故障点。此外，传统网络架构的核心交换机或路由器中，通常搭载了完整功能的软件算法，使得设备可以适应各类场景，但通常一个设备的运行周期之内，设备并不会频繁更换拓扑连接，也不会随意中断运营业务，这就造成了设备功能臃肿与制造高成本之间的矛盾。综上所述，传统网络设备的软件代码量庞大，生态闭塞、难以快速增加新的功能，灵活性低、不具有可编程性，因此急需要一种新的网络体系架构，来解决传统网络可扩展性差的不足之处。

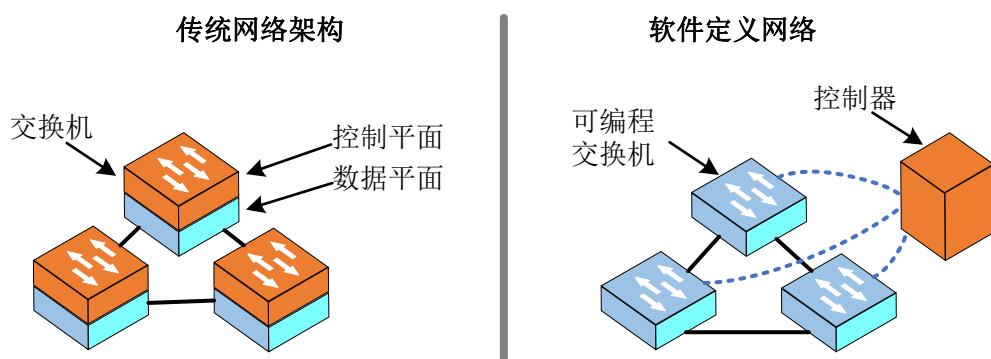


图 1-1 传统网络与软件定义网络架构比较

作为可编程网络的重要组成部分，软件定义网络（SDN）技术（图 1-1 右侧所示）将网络设备中的 SDN 将数据平面和控制平面解耦^[9, 16]。在数据平面上，对数据包的处理统一做查找-转发（Match-Action）抽象。控制平面负责建立网络拓扑，控制并下发流表。这样所有的数据包转发行为都由控制平面的软件逻辑管理，数据平面的设计将变得统一并且简单。由于软件具有强大的灵活性以及开发的敏捷性，SDN 大大加速了网络创

新和智能化管理的进程。OpenFlow^[9] 协议已经是数据平面和控制平面的通讯桥梁的事实标准，并由安全通道（Secure Channel）构成。SDN 网络的数据平面趋于统一化、简单化，是用于完成计算机之间通信数据包的匹配、修改、传送、转发的软硬件设备，厂商只需生产符合 OpenFlow 控制协议的白盒交换机，则设备可以被编程为适用与各种场景的状态。

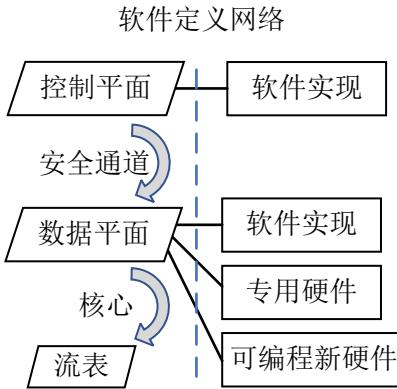


图 1-2 软件定义网络实现方案设计

数据平面的可编程性核心依赖于流表及其配置操作，网络管理员拥有对数据平面的各个特性做快速个性化定制的能力。网络的控制平面维护全网视野、调度、配置针对流的转发条目。运行在控制平面内的各种应用构成了其全部功能。当前 SDN 网络系统设计架构如图1-2所示，在不同场景下，网络对数据通信的需求千差万别，根据应用场景流量大小不同、处理过程复杂程度来设计选取数据平面的实现方案，目前主要有软件、专用硬件和新型可编程硬件等实现方案。两者在功能上都能够对数据包做一系列处理，包括匹配、查找、统计、传送、转发和安全校验等，其中“流表”是实现数据平面功能的核心函数（器件）。数据平面包含一个可以与远端控制器沟通的代理机构，这部分功能着重于协议通信以及安全通道信息加解密，主要由轻量级处理器构成。

硬件交换机之于软件交换机的主要区别在于处理数据包的性能以及交换容量。数据包处理性能主要参数为吞吐量（字节每秒）和包吞吐量（包每秒）。目前基于软件实现的数据平面性能可达到 60Gbps/60Mpps^[17]。当数据包处理复杂度（操作步骤数量）增加时，软件交换机的性能会大幅下降。专用硬件交换机有接口数目多、交换容量大的特点，一般能满足 64 个 100G 端口的总交换容量。而且硬件交换机转发时延低，性能与数据包处理步骤数目几乎无关，稳定性良好。在核心网络和高性能网关等领域主要使用基于硬件的交换设备，但成熟设备功能固定、更换成本高，如果需要更新网络功能，几乎无所适从。所以目前在数据中心网络的 NFV（网络功能虚拟化）等场景内，软件交换机依然占据很大份额。由于软件交换机灵活性高，开发人员能够快速迭代部署新功能，且传统单机通信速率需求不高，软件交换机尚能满足需求。但随着人工智能、5G 领域的发展，数据中心网络内通信容量需求快速增长，转发时延收紧，软件交换机性能瓶颈凸显。运营商不得不为网络任务大量堆叠服务器。本文主要侧重于研究服务器和核心交换网络中，如何使用可编程硬件来大大缓解网络性能瓶颈。针对控制平面，

本文使用 SDN 全局优化的思想，实现对网络中的瓶颈资源（如流表资源）的可扩展性并提升通信协议的安全性。

随着创新需求的进一步提高，只有让底层硬件拥有灵活的可配置能力才能满足目前行业变革的需求，因此研究人员提出了数据包处理编程协议无关（Programming Protocol-Independent Packet Processors, P4^[18]）概念。如果说 SDN 给出了控制层的全局视野，那么协议无关可编程的数据平面给出了设备层的全局视野。SDN 已经将数据平面高度抽象，操作人员可以灵活地定义数据流，以及对这种流进行怎样的操作。但是在数据平面内数据包头的匹配域却是预先规划好的。固有转发平面的设计思想会引起如下两个问题：其一，添加新特性需要跟业界讨论、以及等待很长的设备研发时间；其二，在数据平面内固化现实中可能出现的每一个网络协议字段造成宝贵计算资源的巨大浪费。满足新阶段的网络创新需要具有比 SDN 概念更好的灵活性、动态性。因此斯坦福大学提出了 P4^[18] 编程语言框架，这种语言有能力重新定义数据平面的包解析模式。P4 源代码通过前端编译器编译为中间表示层代码，这个编译过程将提出源代码中的语义逻辑。之后需要根据不同的目标器件再进行后端编译，这个过程最终会生成目标器件对应的机器码，硬件可直接读取。目前 P4 的目标设备已经有基于 ASIC 的交换芯片、CPU、GPU 和 FPGA 等多种实现。

P4 扩展了可编程抽象的灵活性，包括可编程的包头域抽取器，以及可编程的多级流表。P4 同时规范了一种编程语言^[18]，它可以控制数据平面对数据包的任意解析行为，也可以自由配置查找表的数据位宽和多级流表之间的查找流水线^[19-20]。这种更高阶的硬件编程模型使交换设备更加透明，设备与网络协议解绑。除此之外，网络端到端的大带宽、低时延需求引申出了网络功能硬件卸载、网络随路计算等概念，这也进一步促使网络面向全面可编程化方向发展。协议无关处理器从诞生至今已经覆盖了广泛的网络应用场景：1) 替代传统网元（服务负载均衡^[21]、安全控制^[22]、流量控制^[23]、测量^[24]），使云网络自身成为一个软硬任务分配均衡且可编程的系统。2) 增加网络随路专用功能，如键值查询（key-value store）^[25]。旨利用网络高速以及可编程交换机特点，使特殊功能的性能大幅提升。

P4 规范的网络可编程数据平面可编程性依然比较弱，只能自主设定包头域以及匹配表，无法增加其他适应性计算。在一些网络延时要求严苛的场景，网络随路计算概念应运而生，网络可编程性差也成了制约网络发展的关键因素。编程灵活性，与吞吐性能是天平的两端，由于网络巨大的性能需求，很难同时获得很高的可编程性。在此之外，也有很多特性是 P4 架构无法实现的：1) 包头长度有限，目前数据平面可编程的流水仅仅局限于处理宽度受限的包头。2) 非图灵完全，只有有限个数的固化的“执行器”。3) 没有讨论包调度问题。4) 无法对数据包进行可编程的带状态处理。而这些问题目前被认为是因为追求高性能而带来的设计折中^[26-27]。

1.1.3 可编程网络的关键问题

复杂化的网络架构以及快速增长的数据包吞吐容量，对网络的管理方式以及可编程的配置能力都提出了巨大挑战。本文主要围绕计算机网络可编程化问题中的处理性能增强以及可编程灵活性提升等两个子问题进行了深入分析，可编程网络由软件定义网络概念引出，其将网络分为了控制平面与数据平面。在优化数据平面处理灵活度时，又发展出了数据包协议无关处理器等概念，其可针对不同协议数据包的处理需求，灵活地在线升级数据平面的功能，从而追求提升数据平面的可用性。针对这两大研究领域，现有的可编程网络设计领域主要面临了以下三方面的问题：

- **网络数据平面的运算资源问题：**转发设备中的流表是控制平面表达控制灵活性的关键运算资源。为了快速处理数据包，数据平面内部一般使用特殊的运算资源来降低每个数据包的处理耗时，例如使用快速存储器或硬件查找表，而这类专用运算资源一般会耗费较大的存储资源和制造成本。作为高性能网络内设备核心资源的流表因造价高、容量小导致网络内极易产生流表溢出等现象。随着网络数据包吞吐容量以及 SDN 网络对流表消耗量的快速提升，核心运算资源数量有限与其分配的供需矛盾问题日益凸显。
- **网络数据平面可编程能力问题：**虽然硬件交换机在网络性能方面大幅超越基于通用软件服务器的数据平面，但是基于白盒交换机和数据平面可编程专用芯片的编程能力受限。目前实现灵活网络功能主要依赖于全可编程平台构建的数据平面，包括基于 CPU 的通用软件处理平台、网络处理器以及可编程逻辑器件（FPGA）等。但真实环境中的包处理过程非常繁琐，现阶段可编程数据平面依然有着灵活性与性能之间的矛盾，全可编程平台处理效能极低，而且目前基于 CPU 的处理平台性能增长放缓，这也限制未来对网络功能处理能力的提升。
- **网络数据平面的编程抽象问题：**已有的网络数据平面编程方式主要依赖于纯软件或纯硬件的全可编程平台，端侧网络主要由基于 CPU 的主机软件以及网卡构成。CPU 已经成为主机网络通信速率的瓶颈，全可编程平台拥有极高的可编程灵活性，却缺乏与之匹配的性能支持。提升系统整体处理能力，既要对软件做优化也需要对硬件做优化，对于数据中心内众多网络功能（网络测量，拥塞控制，入侵检测等）优化工作需要进行复杂的软硬件协同设计，开发难度大、周期长。因此，急需解决针对这一系列网络功能高效的编程方法。

针对以上可编程网络系统设计领域所面临的困难和挑战，需提出行之有效的解决方案以提高目前可编程网络系统的可用性。本文从网络系统的管理控制层、核心交换层以及终端接入层出发，提出了软件硬件相结合、并利用可编程硬件与传统数据平面相结合的思路来解决相应挑战。下一小节综述了国内外网络基础设施技术的演进，主要分析其技术特征和局限短板，重点关注了现阶段实际情况下 SDN 可编程数据平面的灵活性与性能矛盾点，以及 SDN 网络架构下网络设备硬件资源匮乏的现状。通过分析目前已有的工作内容，凝练出目前研究现状的不足之处以及可以改进的关键问题，为本文研究工作指明了方向和意义所在。

1.2 网络数据平面的研究现状

近年来，网络数据平面呈现井喷式发展，对网络设备性能的提升带来极大需求以及挑战。IDC 报告称，2019 上半年中国公有云服务整体市场（IaaS/PaaS/SaaS）达到 54.2 亿美元，并预计在未来 5 年间内以年均复合 46% 的速度快速增长^[28-29]。数据中心内服务器计算力呈现异构化趋势，GPU，AI 芯片，FPGA 等使用非通用类型指令集和特殊体系架构计算单元已成为目前分布式计算领域的热点话题。现在超大型数据中心一般可容纳数十万台终端服务器，内部网络链接数量多、拓扑规模大、传送海量数据，这使得现有的网络变得异常复杂^[6, 9, 30-31]，同时新的数据包类型层出不穷也使传统网络显得不够灵活。

在计算机网络引入“可编程概念”之前，网络设备的核心运算资源是流表，由于传统网络对流量细粒度控制能力比较弱，因而数据平面运算资源出现瓶颈的概率很小。在新型的具有高度灵活性的可编程网络出现之后，数据平面对于核心运算资源的需求显著上升，同时也遇到了处理性能的瓶颈。本结将简要从解决以下三个方面的问题来回顾网络数据平面可编程研究的现状，分别是网络数据平面的运算资源问题、编程能力问题、以及编程抽象问题，并分析目前研究成果的不足之处以及值得继续关注的关键问题。

1.2.1 网络数据平面的运算资源问题

1) 软件定义网络控制平面资源

软件定义网络（SDN）^[9] 的诸多优势是由于网络结构逻辑分层带来的独立可支配的控制平面，控制平面上可运行足够复杂且编程灵活的网络应用。数据平面与控制平面之间的交互称为“南向接口”，目前南向接口的事实标准是 2008 年由斯坦福大学提出的 OpenFlow 协议。OpenFlow 协议由最开始的 OpenFlow1.0，快速发展到现在的 OpenFlow1.6。几年时间，OpenFlow 协议已经逐步完善到网络的各个细分领域：流量调度^[32-33]，光适配^[34]，广域网^[10, 35]，超转发^{②[36]} 等。SDN 将网络业务抽象为网络操作系统（控制面）上的不同应用程序。如图1-3于控制平面与数据平面二者为远距传输，通信成本相对增大，主要体现在：第一，SDN 强调网络的快速变化，然而控制器对数据平面的控制都依赖于容易成为瓶颈的安全通道。安全通道一般通信速率较低，而且控制信令传输延迟大。这与快速变化的网络结构成为矛盾。第二，成为中间瓶颈的安全通道容易承载来自内部、外部的大流量而容易遭受攻击。

SDN 的控制平面包括一个符合南向接口协议的网络操作系统（网络 OS），以及运行在其上的众多应用。SDN 网络操作系统与下层数据平面通过安全通道相连，数据平面中存在需要快速变化的流表项信息。网络操作系统基本的任务是管理与配置，其中管理包括：发现交换机、发现拓扑、发现端口、故障测量等任务；配置包括：流表配置、执行集配置，组表以及限速表等配置。在支持数据平面可编程（PISA）的交换机中还会有包头解析器配置和数据平面逻辑块的配置。由于网络动态性强的特征，上述配置

^② Super Packet Transport Network, SPTN。一种硬件功能组件可分解的高效可编程网络框架。

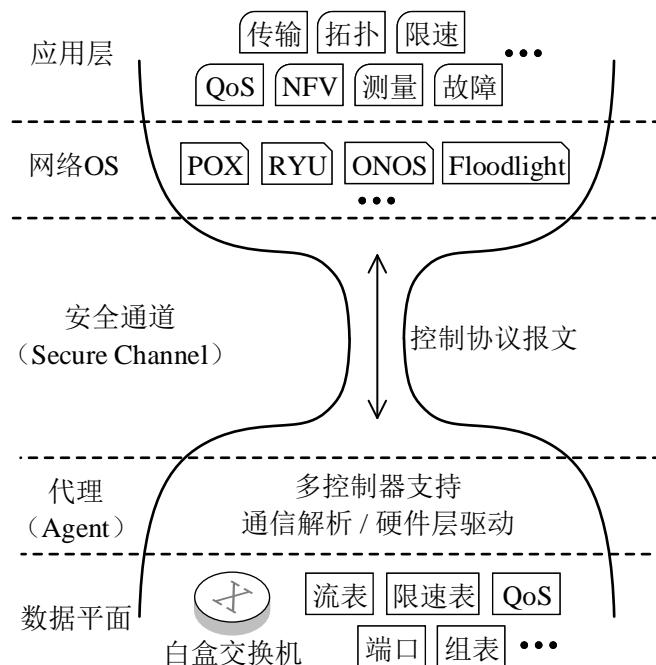


图 1-3 SDN 架构的瘦腰问题

内容均可能快速发生。

为减小集中式的控制平面面临的性能不足的风险，文献 [37-39] 将一部分控制平面的路由算法放在了数据平面，文献 [40-41] 通过多控制器的方法扩展控制平面的处理性能，ALO[42] 优化了多控制器之间的分层结构关系，使得控制信令在多控制器之间更易于管理。针对于流表配置任务，如处理新流到达，SDN 有两种策略，一种是动态响应（Reactive）^[43]。Reactive 的核心思想是被动实时处理数据平面内出现的新流量。交换机如果发现这是一条无法匹配到结果的流，那么交换机会将次信息上报控制器，控制器认证、处理后将新的流表项下发到数据平面，从而完成此流后续转发。Reactive 的缺点就是控制平面与数据平面之间信息交互频繁，对安全通道造成很大压力，若遇到瞬时流量突发（burst），还有可能会耗尽控制器的能力资源，使数据面服务中断。另一种是规划响应（Proactive）^[44]。Proactive 的核心思想是控制平面根据网络拓扑、传输任务意图，提前将所有可能出现的流量信息全都下发到控制平面。这样可以避免后续实时配置过程中的不确定性，减小安全通道遭受大流量冲击的概率。但 Proactive 的缺点也很明显，他需要大容量的硬件流表转发表来预先安装可能还用不到的流表项，另外它还与流表项更替的一般思路“最近最少使用替换”（LRU）算法相冲突。

2) 软件定义网络数据平面核心处理部件

交换机的本质工作是找到对某一数据包的处理方式，并执行这种处理，可统一称之为查找和执行。查找在计算机学科内是一类最基本的问题，例如存储器就是一种典型的查找系统。总线输入数据地址（address），存储器可以返回对应地址上的数据（data）。在网络领域，输入的数据地址其实就是匹配域的值（key），返回的数据就是待处理的操作数（action）。这种过程也可以抽象为解决“key-value”的对应问题。操作数就对应着

对数据包的具体执行动作，数据包在之后的流水线内可以被执行机构按操作数值进行处理。

传统的表项查找方法是基于软件查找方法^[17]，其特点是表项容量较大，查找速度无法稳定高速运行。线型查找法需要遍历表中的所有表项；二叉树查找法需要遍历树中大多数节点，而且查找速度受树的深度影响较大。目前基于 Linux 内核的软件查找性能只有 1Mpps^[45-46]（64 字节最小包 1Gbps 吞吐）左右^③，是远不能满足核心路由器的处理需求的。数据平面交换机设备为支持大量流快速查找，一般需要使用基于 RAM/CAM 等硬件的快速存储器。第一，基于 RAM 的精确匹配查找。RAM 是最简单的一类流表查找方法。如图1-4所示，在初始化配置流表项时，包头域的值作为地址，操作数作为内容，存储到 RAM 内。查找时先从包头提取出匹配域的值（key），然后读出以 key 为地址位对应的数据即可得到操作数。一般 RAM 查找的时间复杂度只有 $O(1)$ ，但如果待查找的匹配域过宽，则会消耗掉一个很大的存储空间。第二，基于 CAM 的内容地址

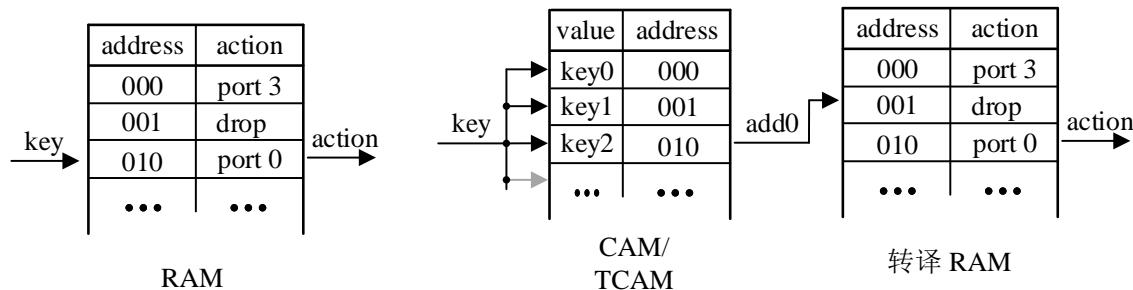


图 1-4 基于 RAM 的包头域查找过程

图 1-5 基于 CAM/TCAM 的包头域查找过程

查找。由于 RAM 查找法对于长匹配域无法实现资源优化，因而提出一种只跟内容数目相关的硬件查找方法（CAM）。如图1-5所示，在配置 CAM 时，将待查找 key 作为内容存储在 CAM 中，与 RAM 类似，每一个 key 都会对应到一个地址位置上。CAM 的输入内容是 key，当 CAM 接收到查找请求后，首先将 key 同步广播到每一个内容存储单元内，同时进行比较，如果与之前存储值相同，则会返回匹配成功，同时返回此单元内数值所对应的地址位置（addr0）。因此 CAM 表增加了芯片控制逻辑，从而同时保证了低处理时间以及小存储资源占用。第三，基于 TCAM 的三态内容地址查找。与 CAM 类似，都是讲 key 存到芯片存储单元内。TCAM 可以支持任意位的掩码查找，也就是说可以在匹配时对某些 bit 位设置“不关心”状态，所有不关心 bits 都可以认为是匹配成功，TCAM 匹配的时间复杂度也是 $O(1)$ 。

流表在可编程数据平面内担任越来越重要的作用。在分离的数据平面与控制平面的体系架构下，控制器所有灵活的可编程算法都需要向流表发出指令，并内写入对应的表项来定义交换机的动作。P4 处理器利用可编程流表以及 offset+length（偏移 + 长度）来灵活定义流表结构^[18]，作为控制平面和数据平面之间沟通的核心纽带，流表最终实现了可编程的数据包转发动作。流表是数据平面内的关键性资源，由于高速交换机内基于硬件的流表容量一直受到电路半导体面积的限制，所以流表也一直面临可扩

^③ 指单个 CPU 核心，如果多核并行性能可进一步提升，但一般只能达到亚线性增长。

展性差的问题。只有数据平面内的流表空间足够大并且充分利用，才能体现更复杂的可编程语义，以及可编程动作集。

为了提高数据平面流表的容量，CompactTCAM[47] 提出一种宽、窄包头域的等效替换方式，压缩了已占用流表容量^[48]，从而减小流表宽度的需求。由于更改了其他公共包头域的编码状态，这个机制适用于内网，无法直接用于大容量需求的骨干网广域和普通数据中心网络。工作 uFlow[49] 发现网络中小流可以经由控制器直接转发到目的地，从而不占用网络内流表存储。其实大流的溢出才是最危险的，而且控制平面与数据平面混合会导致控制平面遭受 DDoS 攻击，并且现有工作都是从减小开销的角度去解决问题。然而，这类解决方案都没有解决当流表资源不足之后，如何应对处理性能快速衰减的问题。

综上所述，目前网络数据平面的运算资源问题包括了两个子问题。首先由集中式的控制平面所引发的网络功能计算量负载大，因此当数据平面对控制平面请求数量过多容易导致控制平面无法响应。其次，高性能的数据平面核心运算资源流表容量稀缺，这使得数据平面向控制器南向接口请求过于频繁而导致网络服务中断。目前数据平面与控制平面的交互方式遵守 OpenFlow 协议，为了做到最好的兼容，此框架限制了南向接口的大幅度修改，这使得服务中断问题变得难以求解。

1.2.2 网络数据平面的编程能力问题

1) 网络数据平面的灵活性

网络对于业务的基本价值是网络实现了数据在计算机之间的任意传输。在早期一个简单的星型拓扑中，一个路由器其实就是一台普通计算，这在侧面也体现出软件作为网络实施载体的特点：“灵活性”。即：对于处理并实现一个新兴事物，软件可以发挥其巨大的灵活性优势，使其可以作为一种预期功能的手段。后期随着数字信息交互的需求和价值越来越大，研究重点开始关注在如何实现快速的包交换、路由查找，并提出各种快速交换的数据结构：Cache 优化、哈希表、Radix Tree(树查找) 等。CPU 和存储每 18 月性能翻番，网络设备的性能也顺势而上，网络处理从单 CPU 向多 CPU 并行^[50-51]，向分布式存储 cache 结构进行了扩展^[52-53]。到 2011 年底，我国互联网入户带宽平均接近 20Mbps^[54]。从最初 14.4Kb 的拨号上网，网络容量的发展几乎是以每 18 个月翻 10 倍的速度在增长。在数百兆的路由性能要求下用软件作为转发设备基础比较合适，但如果核心网要升级到 1G 或数十 G 以上更高的带宽就会面临技术、成本等多方面的瓶颈。

虽然构建在软件架构之上的网络是最具有“灵活性”指标的，但是数据包交换对于 CPU 来讲是一种很累的工作。虽然数据包转发算法既简单、又高效，但面对无穷无尽的任务量，依靠指令集的软件转发架构存在访存效率差、CPU 无法批处理等劣势。这时研究人员抛弃了基于指令集的软件架构，开始思考基于专用硬件电路（Application-specific integrated circuit, ASIC）的数据包处理模型。数据平面可编程概念引发了众多新技术，并为解决不同问题所提出的创新实践。当设备处理接收进来的每个数据包时，数据平面

是网络当中最关键的环节。通常需要用到专用的硬件设施，或者经过复杂优化后的软件加速方案。在硬件方面，数据平面可以在 ASIC^[19, 55-59], FPGA^[60-67], 网络处理器（NP）^[68-70]，外挂有三态内容地址查找器件（Ternary Content Addressable Memory, TCAM）的系统^[71]，在软件方面有基于快速包分类算法^[72-74]的在 CPU 系统上实现^[17, 75-85]。另外还有一类基于其他异构外设的网络数据包处理机制例如 GPU，但 GPU 善于做批处理，对流式网络数据适配不佳，导致存在性能瓶颈^[86-88]。从 2008 年提出软件定义网络，由于真实环境性能的需求，业界从未间断地开发基于硬件的可编程数据平面。在 P4 概念提出来之前，也有类似于半 P4 的混合型可编程数据平面，受限于设计架构，他们对于短长度域可以实现任意匹配，基本可实现常见协议的数据平面编程，但不能够有效支持宽域^[55]。

2) 数据包协议无关可编程芯片架构

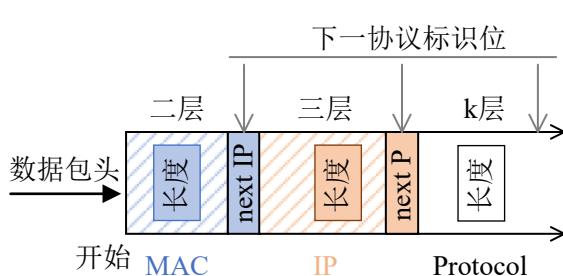


图 1-6 数据包包头结构

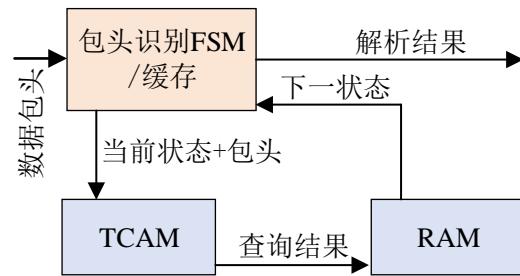


图 1-7 基于 ASIC 的可编程包头协议解析器架构

可编程包头解析器可以实现数据平面对新协议的可编程性。交换机中的查找表必须查找固定协议字段，所以每个交换机都会包括一个包头解析器，它能够标识当前数据包包头的协议名称。如图1-6所示，数据包是一组串行数据，从数据包的开始位置起，包头协议依次串行排列。每个协议段长度固定，每个协议末尾会有标识码标明下一个协议字段名称。本协议字段的长度一般都存储在协议字段内部。图1-7所示为一种基于 ASIC 的可编程包头协议解析模型^[89]，包头中的协议字段是一种有向图关系，图的节点代表协议，向量字段代表转移关系。一个包的包头不一定包含图中所有的节点和对应关系，但当数据包到来时，包头所包含的协议只能是有向图中的唯一一种路径。一般使用有限状态机（FSM）就可以提取出包头内的所有协议字段。FSM 内部存储完整的图关系，只要按照图对应关系来给包头不同字段打上协议名称标签即可。可见为了完成可编程的包头解析需要预先储备大量的查表资源，并且基于此的包处理器流水线的可编程能力是有限的^[18]，依然不具备例如计算，状态转发，可编程触发等功能。

综上可见，目前最具有“高灵活性”的软件网络实现方案已经遇到了严重的性能瓶颈，而基于 ASIC 的网络数据平面编程能力仍然受到比较大的限制。网络数据平面的性能与可编程能力是一对看似矛盾的指标，目前如何提高基于硬件的网络处理器的可编程能力在学术界的研究成果还比较少。

1.2.3 网络数据平面的编程抽象问题

1) 网络数据平面的全可编程抽象

利用智能网卡来卸载操作系统内的网络功能，以期望获得比 CPU 更好的效能，同时还可以兼顾网络设计中不断变化的革新需求。智能网卡也称可编程网卡，相比于普通网卡：智能网卡不但可以完成网卡最基本的作用（主机与网络间通信），还应该有如下特征：输入输出多队列、TCP 卸载、流量整形、规则过滤、虚拟化等^[90]。从而增强一些通用场景下的网络性能：带宽扩容、优化 QoS（Quality of Service，服务质量）、降低 CPU 利用率、降低通信时延等。随着时间的推移，研究人员还发现如果能够将计算^[91-92]、随路功能聚合^[93-94]、缓存^[95]甚至 AI^[96-97]都卸载到网络上，有能力显著提高分布式应用的处理效率。目前能够支持这种将更复杂计算卸载到网络中的网卡，都要求有基于全可编程抽象的网络处理器或 FPGA 的智能网卡来实现。

基于网络处理器（Network Processor, NP）的数据平面，拥有完全的可编程能力^[98]。NP 芯片内部一般包括基于硬件的拥塞控制、队列调度、QoS 等协处理逻辑，还包括一组并行微码处理器。处理器按任务可分为核心处理器和转发引擎。处理器通过预先编写的微码来控制处理过程和内容。NP 有通用的编程模式，一旦有新的技术或者需求出现，可以通过软件语义重新定义数据平面。值得注意的是 NP 中的众核一般使用数据平面专用精简指令集，为了达到节能与节约面积，例如浮点运算等复杂的处理指令是不支持的。NP 的每个内核处理性能一般较差，NP 的高性能需要结合使用专用外部电路，一旦处理的内容无法映射到专用电路，NP 的性能优势会减弱^[99]。另外，NP 编程开发门槛较高，NP 运行软件无操作系统扶持，并且代码移植性差，开发人员需要深入理解 NP 的处理模型，因此 NP 始终只在一些狭窄的领域空间内发挥作用。

基于 FPGA 的智能网卡拥有更为广阔的编程空间^[67, 100]。FPGA 内部有大量 LUT 门电路，以及分布式片上互联网络，基于此结构的 FPGA 可以实现任何定制化的逻辑电路。文献 [65] 将基于 FPGA 的智能网卡数据通路模型抽象化，利用高层次软件来编译硬件功能，可屏蔽底层数字电路设计细节，降低开发难度。FPGA 可以方便地移植程序，设计人员可以将 HDL 代码打包成 IP 核，只要按照规定好的输入输出接口位宽和时序就可以任意复用。在设计电路模组时，一般会使用标准的总线接口来连接不同的功能模块，以增强开发的灵活性。如今 FPGA 厂商也会在 FPGA 中加入专用功能电路来增加芯片集成度、增强 FPGA 的处理某些任务时的性能，如 ARM 核、分布式 DSP 核、PCIe 收发器、分布式片上存储。

2) 专用领域系统的可编程抽象

为应对日益复杂的网络功能的可编程扩展，网络功能可编程需求增加。可编程系统提供专用领域所特有的特性是提升网络功能开发效率的重要手段。高级的专用领域可编程抽象可以大大降低开发人员的编程难度。对于流匹配的可编程思想，SDN 抽象出对流表配置的可编程^[9]，P4 抽象出了对流定义的编程^[18]。在数据中心内，网络的管理很复杂，需同时考虑流量控制，拥塞控制，访问控制，攻击探测等诸多网络功能的实现。

网络测量是实现上文功能的底层核心方法，在性能方面有极大的需求^[101]。流量控制，拥塞控制需要利用网络流量实时速度以及平均包大小等相关信息，访问控制和攻击探测也需要用到流触发的反馈测量机制^[102]。基于被动触发的网络测量系统是一种典型的测量实现技术，测量过程一般分为 1) 捕获，2) 统计，3) 存储三大部分。这三点也是影响测量系统性能的关键因素。目前在可编程数据平面中，支持可编程实现的网络测量技术限制于以基于 CPU 或网络处理器的平台。其性能受到 CPU 架构访存与计算能力的制约，每核心单独处理统计量能力在 10Gbps 以内^[103-105]，相比于通用服务器目前上百 Gbps 的吞吐需求，显然存在比较明显的性能弱势。一方面由于目标流量数目庞大，基于 CPU 架构的快速缓存空间不足，多级存储结构带来复杂的更新机制并且受限于外存储的非连续地址访问速率 ($<1\text{Mpps}^{[45-46, 106]}$)，性能表现力不稳定^[107]。另一方面，CPU 计算架构过于通用化，反而在处理简单重复的重任务时无法发挥优势，尤其针对流式数据处理，CPU 无法建立起可编程的高效率的流水线结构，严重制约了网络功能处理能力的可扩展性^[108]。

综上所述，现代高性能网络正在向软件定义化、数据平面可编程的方向发展。可编程网络架构设计的核心，是构建一种支持映射高层次控制逻辑的数据平面。通过分析目前已有的工作内容，凝练出以下研究现状的不足之处：可编程网络的设计思想目前依然受到核心运算资源不足的限制，以及在工程实践中遇到可编程能力与吞吐性能之间的矛盾等问题，这为本文研究方向指明了道路。下一小节将介绍本文的研究内容主要以解决流表资源问题为基础，利用可编程硬件分别在网络中间节点、主机侧网络中构建了提升可编程性的方法，并形成一套完整的端到端系统。

1.3 论文研究内容与创新点

1.3.1 研究挑战

综合以上研究工作可知，在数据平面的研究中，依然存在由于运算资源不足而导致的网络安全威胁问题，全可编程数据平面性能不足的问题。因此可编程网络数据平面的研究面临以下挑战：

1) 大规模流量场景下的流表资源扩展。研究软件定义网络下流表核心资源的供需关系以及调度问题，对保障交换机转发稳定性以及解决网络数据平面运算资源有限的问题有重要意义。流表作为网络转发设备的核心资源有性能强、容量小等特征^[109]。软件定义网络作为可编程网络的经典范例，其对流量的精细区分更加导致流表容量不足，因而在软件定义网络场景下流表的容量问题变得更加凸显^[47, 110]。由于软件定义网络中建立流转发的行为需要远程控制器的支持，因此一旦流表溢出发生数据平面将会对控制平面产生源源不断的流表项建立查询请求，严重情况下会导致控制通道拥塞，进而影响网络其他控制进程，最终导致网络转发功能失效^[48]。依靠增加流表容量的方式可以降低流表溢出的风险概率，但不会对缓解流表溢出后系统安全问题产生实质性的解决，目前研究领域对问题此尚无成熟的技术以及方法。

2) 数据驱动网络下计算与转发协同优化。研究可编程网络场景下融合计算与通信的协同优化方法, 对解决网络数据平面的可编程能力有限的问题具有重要意义。虽然软件交换机在网络编程能力以及可编程计算领域方面大幅超过硬件交换机, 随着 CPU 处理性能增速放缓, 以及网络数据包吞吐容量迅速增大, 其性能较弱的特点已经成为阻碍可编程网络发展的重要瓶颈^[111-112]。基于 P4 可编程芯片的网络交换机虽然性能足够强大, 但是其可编程空间比较小, 缺乏对复杂计算需求的支持^[65]。如何在硬件环境下解决网络设备灵活性与高性能直接的矛盾, 目前依然是一个开放性问题。

3) 新型编程抽象模型下网络硬件资源受限。研究可编程网络的编程抽象方法, 并保障相应的高性能硬件资源占用约束, 对提升高性能网络功能开发效率以及构建细粒度流量的网络功能具有重要意义。在当前数据中心内, 实现灵活多变的网络功能主要通过服务器软件方式, 虽然软件具有快速部署的特性, 但在实现网络测量领域功能时, 软件处理高时间精度的能力较低^[113]。同时, 面对大量并发触发的网络测量任务, CPU 系统的能效比已经处于劣势^[114-115]。如何利用资源有限的网络数据平面中的可编程硬件, 并在其之上建立一套针对网络测量领域的可编程网络系统, 目前尚待进一步研究。

1.3.2 本文研究内容与创新点

本文以解决流表瓶颈资源问题为基础, 研究在网络数据平面内如何利用可编程硬件来更灵活、更高效地支持多种类别的可编程抽象方法。针对高性能网络以及数据平面可编程领域内的三个挑战, 本文利用可编程硬件分别在网络中间节点、主机侧网络中构建了能够同时提升设备性能以及灵活性的方法, 最终构成一套完整的端到端网络系统。

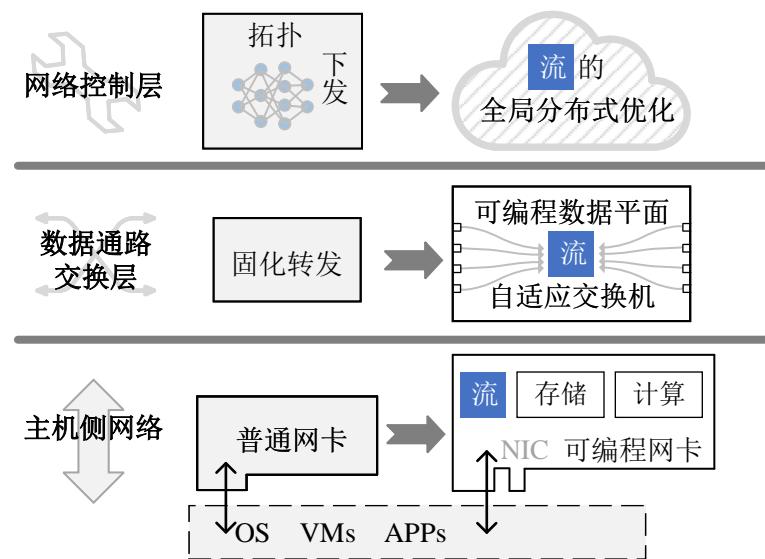


图 1-8 基于可编程硬件的 SDN 数据平面研究框架

如图1-8所示, 本文从网络的三个层次出发, 对可编程网络在每一个层次内做了改进。首先, 流表是基于硬件的高性能数据包转发的核心资源, 在研究内容一中利用软件定义网络控制平面高可配置性, 对数据平面的流表资源进行了全局优化。其次, 利

用可编程硬件与交换芯片结合的异构结构，在研究内容二中设计一种自适应交换结构，优化数据平面网络节点的可编程能力。第三，针对主机侧网络中基于网络测量的网络功能，研究内容三提出了统一的软硬件结合编程抽象方法，利用硬件可编程网卡实现了高性能、高精度以及高效率的网络功能可编程系统。具体研究内容概况如下：

研究内容一：软件定义网络内交换机硬件流表可扩展性研究（第2章）

本文从 SDN 网络全局视野出发，着手解决流表资源匮乏的问题。由可编程网卡和交换机组成的数据平面内，流表资源是提升设备性能的核心，网络数据包的转发动作用于数据平面内查找表的匹配结果，在此基础上交换机内还增加了多种匹配域、多级流表结构，绝大多数平台中都视转发表为最核心以及成本占用最大的模块，软件定义网络（SDN）架构下亦是如此。基于硬件的高性能 TCAM^[116]（三态内容地址查找表）拥有单周期流水、掩码匹配等优秀性能，但昂贵的价格使得用户无法购置容量足够大的表^[117]，因此交换机内极易发生流表溢出的现象。以 OpenFlow 协议为代表，一般规定控制器与交换机之间流表安装流程为 Reactive 模型：交换机收到一条新流首先会上报控制器，随后控制器计算路径并下发流表到数据平面设备。当发生流表溢出现象时，由于 Reactive 模型需要频繁更替活跃流表内容，这会进一步直接引发控制平面和数据平面之间安全通道的消息风暴，否则会造成丢包或服务任务中断等异常现象。第一个研究内容下有两个子问题：如何在维持交换机中原有流表容量的前提下，缓解流表溢出所带来的危害？在保持 SDN 网络平面分离优点的条件下，如何利用其全局化优势高效利用网络设备资源？

创新点：本文分析不同的流量规模和特征，以及系统多模块之间的互联协议，提出一种转发设备节点之间的流表共享机制（Flow Table Sharing, FTS）。

1) 为保障控制平面免于遭受大量突发服务请求，FTS 实现了一种离线转发策略，同时减轻了安全通道的通信压力。实现了在流量突发的情形下，保证数据平面稳定性，降低 SDN 系统控制通道拥塞、失效风险，缓解流表溢出导致转发性能骤降的现象。

2) 为保障数据平面流表资源充足，本文分析并修改了目前 OpenFlow 协议中有关 Table-Miss（流表缺失状态）的处理过程，并修改了 OpenFlow 协议中关于未知数据包上报条件，在数据平面内利用交换机组表进行聚合转发，并且可以方便地退回现有规范。

本文论证了单纯依靠增加流表容量的方案，并不能使流表溢出的概率降低为零，并且能够容易回退、向下兼容现阶段的传统方案。FTS 通过增强数据平面自主性等方法，在流表溢出的情况下，该方案和传统 OpenFlow 协议相比，使 OpenFlow 交换机转发 RTT 延迟和安全通道控制报文风暴数量的优化均达到至少 2 个数量级。

研究内容二：可编程设备增强网络交换节点计算能力的研究（第3章）

本文从网络数据平面处理器的底层设计出发，在现有的大规模集成电路设计能力的基础上，构建可行的网络处理器流水线模型。由于新兴的内容应用（社交，虚拟/增强，混合现实）以及工业网络应用（移动性，大数据，机器学习）使得网络追求高的实

时性、可扩展性。网络设备功能多样性随着数据中心、边缘设备的发展而壮大，因此学界对交换层、核心网场景下快速创建灵活解决方案的需求也愈发强烈。可编程数据平面交换机的三类典型设计架构但目前都存在缺陷：1) 软件交换机性能普遍低下，2) 基于 ASIC 的交换机无法拥有完全可编程性，3) 基于 FPGA 的交换机资源有限，交换性能无法满足业界需求。本文研究内容首先提出一种新型的可编程网络数据平面，采用异构设计的思想，解决上述典型设计的缺陷。因此本文第二个研究问题：如何设计一种同时兼顾转发性能和可编程能力的交换设备？如何利用现有设备优势，对其做最小改动以满足设计？如何实现高资源利用率、高灵活性的高性能硬件可编程数据平面设计方法？

创新点：本文提出一种高性能的数据平面可编程硬件架构，自适应交换结构(Adaptable Switch, AS)。

1) 为同时增强 AS 架构的灵活性和吞吐性能，AS 提出一种旁路数据处理流水线模式。通过 FPGA 与交换芯片联合设计的思想，AS 架构可同时提供 FPGA 的高灵活性与交换芯片的强大性能。论文在前述硬件设计模型的基础上，继续研究硬件逻辑高度并行的性能大规模扩展方法。

2) 为增强可编程硬件内的数据处理吞吐能力，AS 架构设计了一种资源占用低的流量均衡式并行处理模型。

3) 为增强 AS 架构控制平面的处理性能，本文设计了一种高效的启发式流表分配算法。AS 架构解决了 FPGA 性能差与资源少的限制、增强了网络芯片的可编程能力，并且提出了一套部署在硬件上的高资源利用率的并行流水线和流表分配优化算法。

论文设计了 ASIC 面向可编程硬件的扩展接口。交换芯片将数据包头拆分并通过高速数据互联载体发送给 FPGA，利用 FPGA 可重配特性实现完全可编程的包头处理。自适应交换系统在满足全可编程灵活性的条件下，与基于 FPGA 的数据平面相比，将数据包处理性能提升 120 倍以上^④。

研究内容三：端侧网络的网络测量领域内可编程研究（第4章）

本文提出利用基于 FPGA 的智能网卡卸载操作系统内网络测量任务，通过软件配置的方式，令系统硬件支持高性能的网络安全、访问控制、拥塞探测、流量控制等多种场景。目前可编程网络领域缺乏对以上偏向底层的网络功能的统一抽象，开发人员需要独立实现功能细节，重新开发某些通用模块，耗费大量的人力物力。许多网络应用的基础是实现网络测量任务，本文将测量过程分解为三类子操作，1) 捕获；2) 统计分析；3) 流量发送。捕获数据包时需要对每个数据包添加时间戳等数据记录工作，以供流量离线分析以及在线分析使用。在线统计分析对高性能和高实时性的特性要求极高。分析后的流量需要按原始流量顺序与速率再向网络接口发送回去。其中统计分析模块是可编程网络测量系统的核心模块，目前基于 FPGA 硬件可编程网卡同时提供了高性能收发和足够强大的灵活性已经可以满足主机侧网络的性能需求，为更复杂功能的卸载提供了有力支持^[118-119]。如何利用可编程网卡实现高精度、高性能保障和高能源效率

^④ 当前的研究中，主流全可编程目标平台性能约为 60Gbps。

的网络功能硬件卸载？提出测量功能可编程抽象、合理部署和划分任务是本文要解决的问题。

创新点：网络测量是众多网络功能的基础，本文针对目前为了功能应用提出了统一的编程抽象方法。

1) 为提升网络功能的编程灵活性，本文将针对网络测量领域的编程方式抽象为统计、触发模型。上层软件通过灵活调用测量数据来做更高层次的网络测量应用，令系统硬件适用于高性能的网络安全、访问控制、流量控制、拥塞探测等多种场景。

2) 为增强终端可编程网络节点的性能，本文提出了一套基于智能网卡的硬件流水线系统，包括数据包捕获功能、测量系统和发送引擎。本文将其中测量系统操作抽象为基础的包个数统计和数据量缩统计方法。

3) 为增基于测量的可编程网络抽象的可用性，本文提出一种基于硬件的存储压缩的无偏估计算法，在节约 38% 的硬件存储空间情况下，相较于软件的处理方式系统吞吐率提升 8 倍、系统的处理能耗节约 90%。

如图1-9所示，将本文的主要研究内容以及研究成果结构化地展示出来。

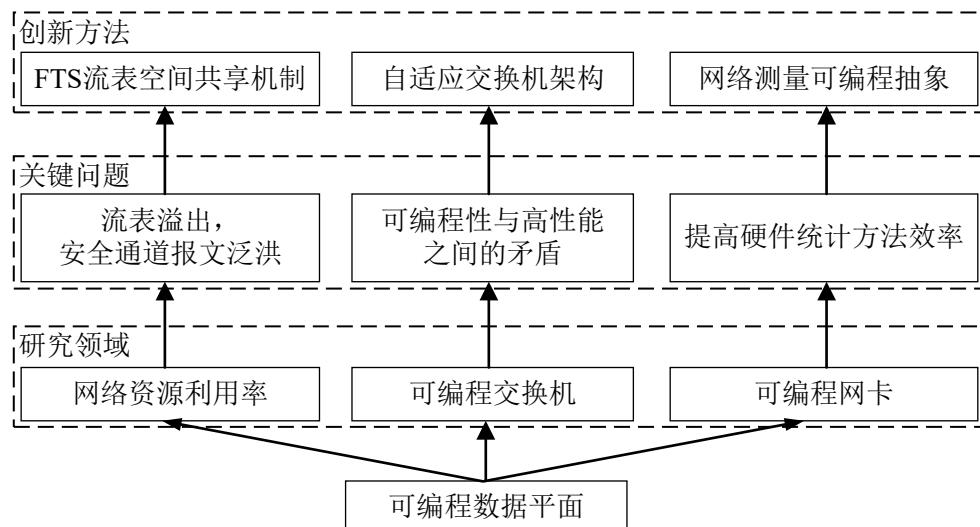


图 1-9 论文主要研究内容以及成果

1.4 论文组织结构

论文共分为五个章节，具体组织结构安排如下：

第1章为绪论，主要讨论研究背景、相关工作综述，分析可编程网络硬件的发展方向、目前研究所面临的挑战以及总述全文。本文接下来三章内容主要针对绪论提出的挑战而做出深入的研究。

第2章主要对网络数据平面内的流表资源做可扩展研究，提出一种全局优化方法来改善流表溢出带来的安全性风险。

第3章论述一种 FPGA 与交换芯片相结合的异构网络数据平面硬件架构，令交换机系统拥有灵活的可编程能力的同时，处理吞吐性也能得到显著提升。

第4章提出了一种高资源利用率的针对网络测量领域编程抽象，通过不同的软件调用，令系统硬件专用处理器适用于高性能的网络安全、访问控制、流量控制、拥塞探测等多种场景。

第5章总结全文工作，并展望未来研究方向。

2 SDN 网络交换机硬件流表可扩展性研究

2.1 引言

本章将通过分析软件定义网络（SDN）通用交换机与控制的控制信令的交互过程，发现目前 SDN 网络由于流表资源缺失而容易引起网络通信失效等问题。软件定义网络的事实标准为 OpenFlow 协议，其将网络解耦为数据平面和控制平面。控制平面内可运行网络操作系统，所有传统的网络转发功能都可以由操作系统内应用来完成。网络管理人员只需要开发网络应用即可管理网络内所有设备的行为，通常应用具有标准的通信接口，可支持各类高级语言，且运行环境单一且标准，屏蔽了过去由于设备数量众多导致的开发流程复杂等问题。OpenFlow 交换机是支持软件定义网络的标准化设备，负责 SDN 网络内数据平面内数据包转发操作。OpenFlow 交换机支持统一的南向协议，并与控制器通信传递配置信息与数据层状态。

转发设备中的流表是控制平面表达控制灵活性的关键器件。目前高速数据包处理设备性能需求可达数十个 Tbps，因此交换机内部需要使用基于硬件的高速缓存（如，TCAM，SRAM 等）来进行查表操作。带有掩码功能的查找表（TCAM）是做 IP 最长前缀查找的高性能核心部件。TCAM 具有单周期吞吐的流水线掩码查找能力使得其功能难以由一般存储器代替。但其价格以及能源消耗都较大，基于 TCAM 的流表容量通常都比较小，且极有溢出的可能。数据包在数据平面中被流表匹配，并得到相应的执行指令，待由后续机构处理。若数据包在流表中没有被成功匹配，则被称作 Table-Miss（未匹配）现象。一般未匹配现象的处理方式为丢包，但在 OpenFlow 协议中支持了 reactive 处理模式，为后续数据包成功处理，将数据包实时发给控制器，等待控制器处理结果并下发新的可匹配此流的表项。

本章从 SDN 网络全局视野出发，着手解决流表资源匮乏的问题。通过分析不同的流量规模和特征，以及系统多模块之间的互联协议，提出一种转发设备节点之间的流表共享机制（Flow Table Sharing, FTS），实现了在流量突发的情形下，保证数据平面稳定性，降低 SDN 系统控制通道失效风险，缓解流表溢出导致转发性能骤降的现象。在之后小节中陆续解答本章研究内容下的两个问题：如何在维持交换机中原有流表容量的前提下，缓解流表溢出所带来的危害？在保持 SDN 网络平面分离优点的条件下，如何利用其全局化优势高效利用网络设备资源？

2.2 问题背景概述

在软件定义网络中，为了适配各种新的包头域，交换机内所需流表的宽度快速增长。相比较于传统 L2/L3 网络，软件定义网络所定义的包头域宽度需要满足所有流量的转发潜在需求，如表2-1所示，最新版的 OpenFlow 协议中规定有 45 个匹配域，总共需要近 64 字节的流表宽度^[120]。在流表容量不变的前提下，包头域宽度的增加就意味着

着深度减小。另外如上文所述，基于硬件的快速查表方案都存在硬件资源消耗大的问题。流表资源数目不足可直接引发诸多网络问题，例如：1) 数据平面内无服务；2) 因快速更替流表项内容导致安全通道内报文数量激增；3) 增大流表容量使设备价格上涨经济效益变差。值得注意的是，第二点问题会耗费大量控制器计算资源，降低安全通道通信带宽，从而进一步产生影响全网络安全的问题。

表 2-1 OpenFlow (OF) 协议字段数量快速增长

版本号	OF1.0	OF1.1	OF1.2	OF1.3	OF1.4	OF1.5	OF1.6
年份	2009	2010	2011	2012	2013	2014	2016
-	12	12	13	39	41	45	45+ ^①

如何缓解这类问题成为当前研究的重点内容。CompactTCAM^[47] 提出一种宽、窄包头域的等效替换方式，压缩了已占用流表容量^[48]，从而减小流表宽度的需求。由于更改了其他公共包头域的编码状态，这个机制适用于内网，无法直接用于大容量需求的骨干网广域和普通数据中心网络。工作 uFlow^[49] 发现网络中小流可以经由控制器直接转发到目的地，从而不占用网络内流表存储。其实大流的溢出才是最危险的，而且控制平面与数据平面混合会导致控制平面遭受 DDoS 攻击，并且现有工作都是从减小开销的角度去解决问题。在本文后续的 2.3.2 小节中证明，流表溢出在现有网络里是必然发生的，目前工作都没有提出如何应对流表溢出后的性能问题。因此如何能够利用 SDN 全局视野以及目前新兴的可编程数据平面来解决上述问题是本文研究的重点。

流表数量充足可以保证更复杂的网络转发功能的编程能力，流表溢出不但影响转发性能，甚至会导致数据平面与控制器通信报文数量爆炸增长，给 SDN 控制器的安全造成隐患^[121]。网络当中流量变化异常迅速，为减小 Table-Miss 事件，交换机须能够进行快速流表更新或大容量查找。拥有无限大容量的查找表可以将所有可能出现的流，都预先存储下来。但显然不现实，因此提高流表更新速率成为交换设备的一个硬指标，目前商用交换机的流表更新速度不超过 10000 条/秒。基于 TCAM 的流表删除和添加操作比较复杂，且 TCAM 有效容量有限，这为设计人员继续增强性能提出了很大难题。

在解决此类问题时，研究员一般从以下两方面着手：(1) 提升流表项数量。当 TCAM 存储容量固定时，假设宽度为 key、深度为 depth，二者的乘积恒定。显然如果可以降低流表项宽度值 key，可以变相增加流表项数量。在数据中心网络场景下，可以通过重新定义包头域宽度来定义流，无需使用现有过宽的包头域。例如工作^[47] 将所有流映射到 16bits 宽度的匹配域，可增加流表项容量 10 倍左右，等数据包离开网关时在将原始包头还原。但这种思路仅仅在内网有效，而应对如今更大流量的骨干网，核心网显然无所适从。(2) 提高转发设备流表的更新速度。表项替换分为两步，首先控制器选择一条已有表项并删除，之后控制器发送并新增一条表项。一些算法可对 TCAM 存储空间进行优化合并，但多个表项之间会产生依赖^[122]。若希望删掉一条逻辑表项，会牵扯大范围的物理存储内容，这增加了修改表项时的操作复杂度，也使得更新表项速度

变慢。若流表项剩余空间足够，新增表项可以直接写入空白区域而不去更改已有内容。还可以在运行时检测流表命中频次，提前探测并删除“死流”，为表项争取最大的可用空间^[123]。存储分级是一种通用的应对策略，例如使用 TCAM+SRAM 的双表转发，可大幅度扩展流表数目。考虑到 SRAM 对带掩码表项资源利用率低，查找速度慢，一般只能将老鼠流存放入 SRAM 空间，实际应用范围比较窄。

以上这类方法的核心思想是增加表项的原有表项空间，其只能拖延溢出的发生时间，解决思路对缓解流表溢出造成危害依然无效。本章节针对前文研究不足，提出一种全网流表资源共享机制（Flow Table Sharing, FTS）。其关键解决问题在于流表发生溢出后，如何缓解控制器计算压力，安全通道消息风暴，以及数据平面性能下降的问题。FTS 修改了经典 Table-Miss 处理方式，引入了数据平面与邻居节点组网机制，共同抵抗高突发流量。当流表溢出时对未匹配流量利用本地特殊配置转发策略进行处理，无需立即请求控制器，并不会影响正常运行的流表项。

2.3 基于流量特征的问题分析

本文通过分析真实流量特征，来证明单纯地依靠增加流表容量无法完全避免流表溢出现象。本文分析 ISP 服务供应商给出的一段真实数据（新西兰，15 分钟，2 千万个数据包），发现增大流表容量的确可以减少流表的更新速率。如图2-1本文使用最近最少使用（Least Recently Used, LRU）替代算法，当设置流表容量为一万条时，流量变化超过交换机更新极限的情况大约占到 10%。若进一步增大到 5 万条，约有 6% 的流量会引发流表溢出。再将流表容量扩大到 25 万条，仍有 4% 的流量无法得到及时地更新。

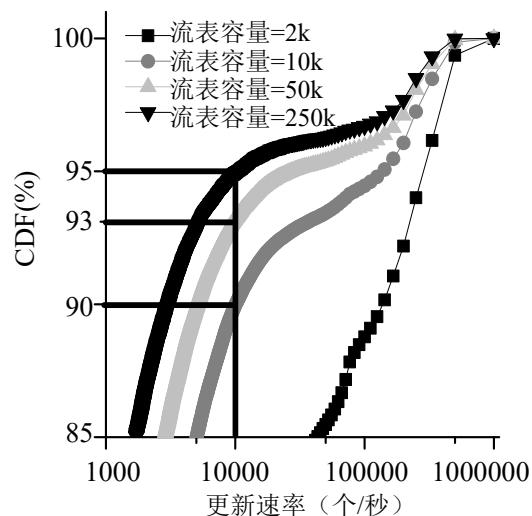


图 2-1 流量更新速率的概率累计分布

根据 Openf 协议，流表最大支持 45 个域（144Bytes），流表资源消耗量大的情况在 OpenFlow 交换机中尤为凸显。面对上述问题，为使 SDN 的优势在广域网间应用，业界不得不使用以及购买性能更强价格更昂贵的交换机。在实际应用中，若交换机流表资源不足，对于最终无法更新的流，交换机会将他们按照 Table-Miss 策略处理。目前的控制策略大都会以 Packet-In 消息的方式上报给控制器。若 Packet-In 消息数目过多，会引

起安全通道传输阻塞以及引发控制器处理阻塞，不但会引发控制器安全问题，且影响到其他服务，会加剧数据平面转发效率下降。

2.3.1 流量细粒度趋势

SDN 的核心功能抽象为控制平面向数据平面安装流表项，OpenFlow 交换机对数据包的处理过程遵照流表项内容进行转发。基于 TCAM 的查找表具有高速，流聚合等优良特性，已经广泛使用在交换机中。然而 TCAM 芯片设计复杂，电路面积巨大，价格高昂（2500 人民币/Mbits），功耗大（15 瓦/Mbits）。因而交换机无法存储全部流量信息。

如今网络已不局限于 best-effort 服务理念，服务提供商和内容分发希望流量有区分地进行服务，以提供不同场景不同服务价格，赚取差异化后的最大利益。此外数据中心和蓬勃发展的虚拟云主机市场也对网络流量细粒度管理提出更多需求^[124-126]。尽管 OpenFlow 协议下流表资源消耗量更大，但市场依然期待让用户得到 SDN 有利技术。越来越多的实例也展示了 SDN 技术的强大魅力以及逐步扩大的影响范围。如何应对大规模部署后日益匮乏的流表项资源，以及解决流表溢出导致的性能极端下降，并且如何高效地利用全网有限资源是亟需解决的问题，也是本章探讨的主要内容。

2.3.2 问题分析

SDN 架构使得控制平面与数据平面在物理上分离，交换机与控制器之间通过专用线路建立连接，通常专用线路称作“安全通道”（Secure Channel）。数据平面的配置内容会直接影响到网络的行为，因此对于交换机的配置需要确保严格的安全性。OpenFlow 协议利用各类消息定义为应用程序接口，数据和功能打包为数据包在安全通道内传递。SDN 对网络的控制因为安全通道传输开销，导致系统控制延迟大于传统网络^[37]。研究人员面对了两个源自于 SDN 架构的挑战：(1) 控制平面—消息数量庞大，控制器一般较为复杂、性能要求极高；(2) 数据平面—流表空间有限，流表溢出易造成性能大幅下降。

第一类问题，研究人员已经提出了分布式多控制器原理^[127]，专用子控制器^[39]，提升控制平面的性能方法。本文主要针对第二类问题，即对数据平面可扩展性问题进行研究，分析其可能造成的严重后果。目前 OpenFlow 交换机有两种实现架构，第一是软件交换机，第二是基于硬件的交换机。二者各有优势，软件交换机通常部署在服务器虚拟机环境内，软件交换机的流表项大多存储于内存空间。服务器内存空间资源通常比较丰富（多达上百 GB），足以保存足够多的规则，拥有资源冗余度。软件交换机应用范围有限，主要适用于交换吞吐较低的场景（10Gbps 以内）^[116]。可将流量较小，流数目丰富的老鼠流交由软件交换机处理。但对于高性能场景（大于 1Tbps），依然须使用基于硬件的 OpenFlow 交换机。

对网络领域进行行为分析，一般使用泊松过程。

定理 2.1：假设数据包的到达服从参数为 λ 的泊松过程， $Flow_number(T)$ 为流数量随时间分布，则必然存在某一时刻，使得容量为 C 的流表充满。即存在某时刻 t 使

得：

$$Flow_number(T) \geq C, \quad T \in (t, t + \Delta t) \quad (2-1)$$

□

证明：假设交换机中流表容量为 C 条，数据流的到达模式为具有参数 λ 的泊松过程，流的生存时间为服从速率参数 μ 的指数分布， N_t 为在 t 时刻流表中的流个数。显然， N_t 是一个增消过程，若到达强度 $\rho > 1$ ， N_t 将无边界增长。因而需满足：

$$(\rho = \frac{\lambda}{C\mu}) < 1 \quad (2-2)$$

N_t 的稳态概率密度分布函数推导如下^[128-129]：

$$\pi_0 = [\sum_{k=0}^{C-1} \frac{(C\rho)^k}{k!} + \frac{(C\rho)^C}{C!} \cdot \frac{1}{1-\rho}]^{-1} \quad (2-3)$$

$$\pi_k = \begin{cases} \pi_0 \frac{(C\rho)^k}{k!}, & 0 < k < C \\ \pi_0 \frac{\rho^k C^C}{C!}, & C \leq k \end{cases} \quad (2-4)$$

其中 π_k 代表流表中已经有 $k (= Flow_number(T))$ 个流表项的概率。参考爱尔兰公式^[128]，流表充满的概率为 $P(C \geq k)$ ：

$$\begin{aligned} P(C \geq k) &= P(C, \lambda/\mu) \\ &= \frac{\left(\frac{(C\rho)^C}{C!}\right) \left(\frac{1}{1-\rho}\right)}{\sum_{k=0}^{C-1} \frac{(C\rho)^k}{k!} + \frac{(C\rho)^C}{C!} \cdot \frac{1}{1-\rho}} \\ &= \frac{1}{1 + (1-\rho) \left(\frac{C!}{(C\rho)^C}\right) \sum_{k=0}^{C-1} \frac{(C\rho)^k}{k!}} \end{aligned} \quad (2-5)$$

由公式2-4中 k 超过 C 时的概率密度函数，以及公式2-5，显然不论 ρ 、 C 、 μ 取何值，总有概率 $P(C, \lambda/\mu)$ ，使得 $Flow_number(T) \geq C$ 。 ■

为得到真实流量特征的统计，本文对广域网（ISP）流量^[130]样例进行了分析，并截取如图2-2所示 30 分钟时间内流数目情况（流分类取前 16bit 目的 IP 地址）。可以发现，平均流数目为 Avr=8420，假设控制器每分钟对全网流量（削峰填谷）实施调度，则最大流表容量需求为 Max_Avr=9120 条。然而在极短暂时刻同时 ($t + \Delta t$) 最多有 k=9663 条

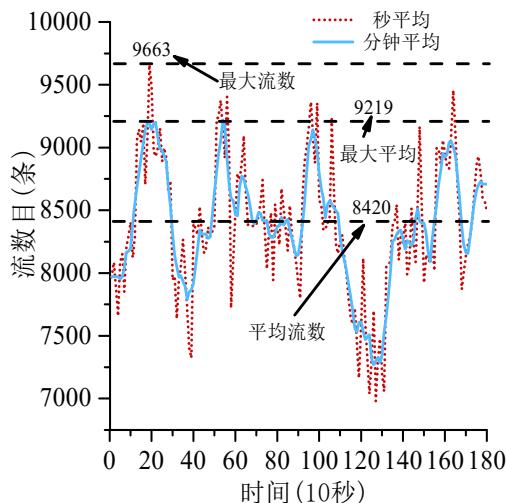


图 2-2 以 10 秒为窗口统计流数目

流同时到达交换机。因此可以计算出流数目超过 Max_Avr 的概率为 8.9%，剩下 91.1% 的时间内，交换机中有 7.6% 的流表是空闲的。网络开发人员应根据已有设备的流表资源来决定调度策略的最大精细程度。根据流表使用比例动态调整流表项分布策略是可行的。为保证应用尽用，使经济效益最大化，则有约束条件：

$$C \leq \text{Max_Avr} \leq k \quad (2-6)$$

$$\text{Avr} \leq C \quad (2-7)$$

公式2-6表示容量 C 若小于最大平均值 Max_Avr 有助于提高流表资源平均使用率。公式2-7保证流表不可被时时刻刻充满的冗余性条件，因此实际平均使用数目须小于 C。在上述两公式的条件下，全局资源中一定有过多的冗余，本文希望减小全网的流表空间浪费率。

数据平面发生流表溢出的概率始终存在，下面分析流表溢出现象对网络通信服务造成的性能危害。OpenFlow1.3 协议规定，当控制器准备向交换机下发流表项时，若交换机流表已经被装满，其会向控制器发送错误报告 <ofp_error_msg, OFPFMFC_TABLE_FULL>，并且交换机端会拒绝安装这条流表项。控制器得到错误返回后，可以选择忽视这条新流，也可以执行流表替换。若：1) 忽视此流，则会造成拒绝服务现象；2) 控制器服务此流，则需先删除某条已有规则，再重新添加此流表项。但如果被删除的已有流表项是活跃流，继而又会造成活跃流中断服务。因此有可能导致数据包传输延迟增大，传输速率降低，发生丢包现象。本章将在2.6章测试流表溢出后通信效果的变化。

下面本文以一个实例具体分析以上论述，图2-3所示为交换机组成的简单拓扑。每交换节点都与控制器相连。假设在某时刻，S2 交换机内流表容量空间被填充满。之后如果再有新流到达，按照目前 OpenFlow 协议并对 S2 交换机分析系统操作流程：(1)

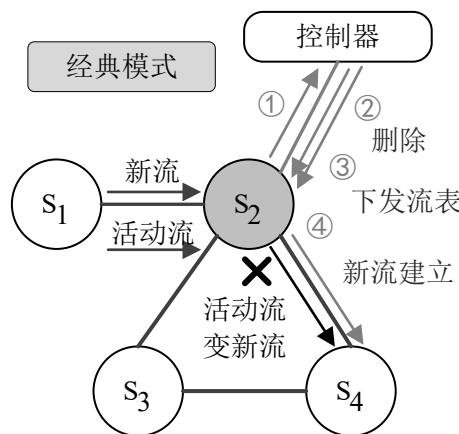


图 2-3 OpenFlow 协议新流处理过程

S2 交换机查找表项未匹配，并触发 Packet-In 消息；(2) 控制器接收 Table-Miss 消息后下发新表项，并且控制器 S2 流表发现已经溢出，选择某条流表项删除；(3) 控制器下发新流到 S2；(4) 对新流转发建立成功。

由于控制器删除的原有流有可能是某活动流 (active flow)，并未超时退出，此流接续数据包会快速到来。然而控制器无法轻易获取数据平面内哪条流是最长时间没有到达的，控制器无法按照最优方式删除活动流。在流表项被删除后，后续数据包会被交换机重新判别为新流。网络控制闭环将循环重复前面的过程。新流频繁到达会导致大量重复控制消息数据包充斥在安全通道内，大量耗费控制器的计算处理资源，且占用安全通道有限通信带宽，同时使交换机服务新流量效率低下。另外一种解决办法：控制器发现流表空间存储已满，不向交换机内替换新流表项。这样虽然保证了已有流量正常转发，但会导致系统放弃对新流的响应。综上，本文面对问题最根本原因是由于流量大小起伏波动，流数量暂时超过交换机容量上线，导致单点转发性能下降，进而影响整体网络服务质量。

2.4 流表共享机制

针对之前两小结节所提出的问题，本文提出了一种全局流表共享机制 (Flow Table Sharing, FTS)，该机制目标为缓解由于流表溢出造成交换机转发能力下降以及暂时无法提供服务的现象。FTS 机制实现两个关键特性优化：1) 允许流表溢出时交换机转发新流；2) 减小安全通道内重复控制消息的数量。

2.4.1 允许流表溢出时交换机转发新流

挑战 1：没有建立流表项的流无法执行转发动作。在经典的<匹配-执行>操作方式，数据包在被匹配成功之后方能得到正确处理。当交换机内没有足够流表空间，针对新流匹配表项无法安装。交换机不会对新流执行任何操作。但本文希望当交换机无对应流表项时允许转发，这与目前模式相矛盾。

解决方案：为交换机增加随机转发特性。与<匹配-执行>不同，随机转发只将流量转发到邻居节点。这种方式没有精确的策略指导，转发出口只要不等于入端口即可

(避免路由环路的产生)。因此这条本被拒绝服务的流，出现在邻居节点内。由于邻居节点与本地节点同时发生溢出的概率很小，这条流会有更大的几率成功地在邻居节点上建立流表项。

2.4.2 减小重复控制消息的数量

挑战 2：由于新流频繁到来，会引发重复的 Packet-In 消息。在 OpenFlow 协议下，交换机收到无匹配规则的流后，会触发 Packet-In 消息机制。Packet-In 消息携带包头信息给控制器，控制器依据此判断处理方法。当流数量超过流表容量时，对交换机而言当前时段内无法处理的数据包都可归为新流。将导致带有相同包信息的控制消息在安全通道内反复传递，占用有效控制信息通信带宽。

解决方案：本文将通配转发引入随机转发模式内。因此对于任意一条无法处理的新流，都会匹配执行。交换机总能保证数据包被处理，从而避免产生重复的 Packet-In 机制。下一节详细论述 FTS 结构的设计与实现。

2.5 基于 OpenFlow 交换机的随机路由策略

为抑制在流表溢出时更新流表而产生的重复控制消息，改善由于新流没有相应<匹配-执行>而信息无法获被转发的现象，本文提出一种<随机转发-流表扩散>方法。流表溢出的节点利用邻居节点内的空闲资源，重新建立从邻居到目的地的转发路径线。

为保证此机制可正常运作，显然如果网络中所有交换节点均可随机转发，必然引起数据风暴或无正确路径。因此假设网络中流表溢出只会发生在随机的少量的节点内。可以利用反证法证明：若网络内大部分结点都存在流表溢出，此情境下的网络必处于大面积拥塞的状态。但从实际观察，绝大多数时段内网络并不会出现大规模拥塞。网络内流表总有足够的冗余度，能满足基本的流量波动。另外，控制器也会实时根据拥塞情况调整网络流量控制粒度。因而流表项的平均使用率只会趋于最大用量，从宏观角度保障不会普遍溢出。

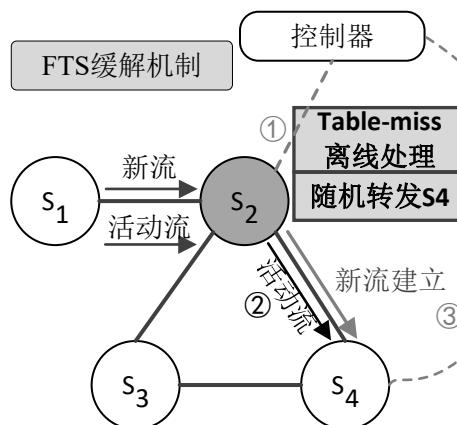


图 2-4 流表溢出后 FTS 对新流的处理流程

图2-4展示了本文提出的 FTS 机制新流的处理方式。假设当交换机 S2 流表溢出，且此时新流到达并匹配为 Table-Miss，交换机不产生 Packet-In 消息。(1) 交换机流表溢

出时，只在本地执行离线策略。通过一定的快速本地计算，得到新流相应的转发动作，从而避免丢弃数据包。其中的快速随机转发，应满足线速需求。(2) 在本例中，新流被转发到 S4 中，且之前建立的活跃流不会受到本地策略的影响，可以持续转发。(3) 新流到达 S4 后，在此邻居节点上重新部署路由。下面的两小节详细说明交换机离线策略及实现方式。

2.5.1 随机路由离线策略设计

本文将交换机中离线随机策略抽象为下图2-5中的形式：

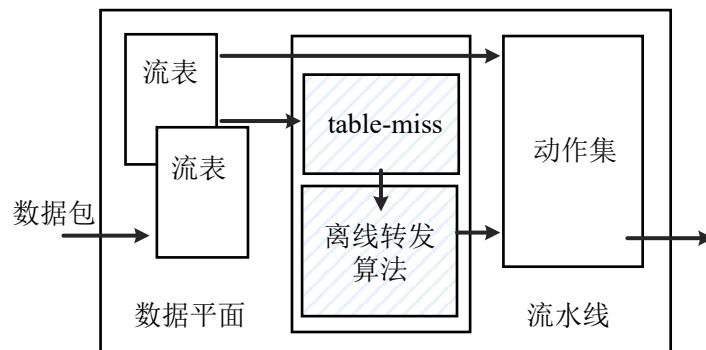


图 2-5 FTS 目标交换机的流水线结构

数据包到达交换机首先执行流表查找，匹配成功后执行已配置的处理操作：例如，计数器更新，指令集执行等。而后判断流表是否处于溢出状态且是否需要触发 Table-Miss 事件。如果没有溢出则直接执行动作集，此处动作集包括触发 Packet-In 消息；若流表溢出，则在本地计算数据包的发出端口。包含 FTS 机制的交换机与 OpenFlow 交换机的区别仅体现在增加触发 Table-Miss 事件时，判断交换机是否发生流表溢出；以及在本地离线处理策略内 out_port 端口计算方法。为支持线速转发，上述算法只能在交换机硬件快速流水线中实施。修改交换机硬件流水线难度较大，但考虑到流水线内的组表，指令集，动作集都等可以灵活配置。因此本文须解决将此算法与 OpenFlow 交换机流水线内拥有可重配特性的组件耦合起来。将本文总体设计总结如下：1. 流表匹配。经过流表匹配后，得到 packet 是否为 Table-Miss。2. 根据 metadata 以及交换机本地判断提供的流表空间使用率，判断是否需要离线计算。3. 流水线中 out_port 的算法计算数据包的发出端口。4. 执行动作集。其中必须满足的限制条件：1. 随机转发的出端口 (out_port) 不可等于入端口 (in_port)，否则产生环路。2. 为满足吞吐速率，out_port 随机算法须简洁高效。限制条件带来以下 2 个硬件实现上的挑战：1. 随机方法。无现成的指令集为随机指定动作集。2. 以目前 OpenFlow 协议实例的硬件流水线无溢出判断。贸然修改交换机流水线不符合 OpenFlow 标准的协议，会导致交换机兼容性差的缺点。本文为应对以上的挑战，选择以组表作为 OpenFlow 交换机实现随机转发的关键部件。

2.5.2 随机路由策略组表方式实现

1) OpenFlow 组表。

OpenFlow 协议规定组表 (Group-Table) 属于交换机流水线内固定功能，组表是多个

动作桶的集合（Action Buckets）。每一个动作桶包括一组可执行动作集和相关参数。组表类型有：ALL, SELECT, INDIRECT, FAST FAILOVER, 四种。本文选择组表的 SELECT 操作类型。SELECT（选择算法）：一个数据包只选定组内的一个桶执行，选取逻辑由用户选择算法指定。例如哈希某些数据包头域，或者基于 Round-Robin 选择。OpenFlow 协议不限定选择算法种类。

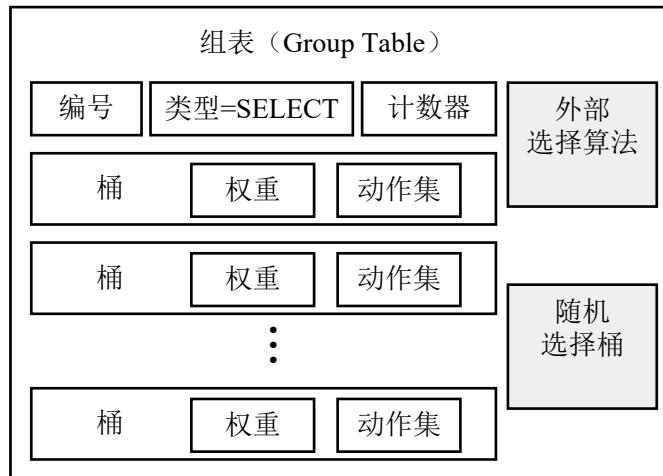


图 2-6 OpenFlow 组表结构

如图2-6所示，组表数据流水线内动作集的一部分。组表将多个动作集内容通过算法融合到一起，给数据包带来更灵活的操作。组表内包括编号字段、组表类型、计数器和桶。不同的组表类型中桶的执行方式也不同。OpenFlow 协议下组表的 SELECT 可以由终端用户自行定义，此选择算法用来从所有桶中选择一个执行，若属某一个选定的桶动作集输出端口失效，逻辑应自动将桶待选范围缩小至剩余可用桶并重新做选择。

某些选择算法已经标准化在硬件组表内，例如，使用哈希算法分类用户配置的包头数据域，或简单的令牌环。本文使用 OpenFlow 组表解决了前文提出的两个挑战，利用随机选择桶算法实现随机，组表天然的又是硬件流水线内的一部分，能够满足线速转发的要求。

下一小节讨论：根据本文提出的组表方案，应该如何设置流表以及组表：(1) 出端口不能等于入端口，(2) 流水线随机算法应简洁高效。

2) 组表桶的设定。

数据包转发出由桶内的动作集用指定，权重表示此桶被选概率的大小。要求向邻居节点随机转发，则桶内的动作集内转发出口 (out_port) 的值应该对应自己本机的邻居节点。显然，数据包出端口不会包括入端。可选择两种方案解决：(1) 结合流表进行二级匹配，区分并剔除源端口；(2) 开发额外组表选择算法剔除入端口。

(1) 结合流表二级匹配。流表项的匹配域为 in_port，设置对应组表的 group_id = in_port。此组表内的桶不含转发到 PORT=in_port 的动作集。之后由令牌环算法实现简单高效的随机动作。需要将此流表项配置成最低优先级，以保证此流无其他流表项可匹配。例如，交换机有 N 个邻居，则需要在流表内添加 N 条表项，再配置相应的 N 个

Input: in_port
Output: out_port

```

1 while New packet arrival do
2   if Not overflow then
3     out_port = CONTROLLER
4   else
5     out_port = (out_port + 1) mod N
6     if out_port == in_port then
7       out_port = (out_port + 1) mod N
8 return out_port

```

算法 2-1 组表外部选择算法

组表，每个组表内包含 $N-1$ 个 buckets；空间复杂度为 $O(N^2)$ 。另外须考虑流表溢出的判断：第一，本文可利用控制器监控流表使用情况，一旦溢出发生，控制器再向交换机内填写 N 个扩散流表项，但是这会造成较大控制延迟。第二，可以利用外部选择算法，在交换机内部判断，控制器须将 N 个扩散流表初始化到交换机内。

(2) 采用额外组表选择算法。扩充组表内的选择算法，则只需要使用一个组表，组表内动作集包括向所有的 N 个邻居、和 1 个控制器（CONTROLLER）端口转发。若算法判断交换机流表没发生溢出，直接选择转发向 CONTROLLER 的端口。在溢出的情况下，out_port 选择算法结果需要同 in_port 做比较。随机算法可以用简单令牌环实现，一旦发现 out_port 与 in_port 相同则选择下一个桶直接执行即可。方法总结 1：空间复杂度大：其中流表占用 $O(N)$ 、组表占用 $O(N^2)$ ；初始化复杂，但是无需用户修改组表内的固定的选择算法。方法总结 2：空间复杂度低：流表占用 $O(1)$ ，组表占用 $O(N)$ ，初始化无延迟，但是需要自行开发组表内的选择算法。由于 OpenFlow 协议对于 SELECT 算法并无强加限制，允许 SELECT 算法由交换机自行构成。因此本文更倾向于使用第二种方案，制定一种可扩展性强的 SELECT 算法（2-1）。

如图2-7(a) 所示，方形框内表示 OpenFlow 协议所规定的组表结构，包括了 1. 组表编号，2. 类型，3. 组表计数器，4. 一个（或多个）桶，5. 随机选择算法，6. 外部选择算法。默认的随机选择算法是基于硬件的令牌环算法。

如图2-7(b) 所示，从逻辑上外部选择算法是 Table-Miss 处理的延伸。因为桶内的权重可被设为 1，表示可被作为 out-port 的候选对象。若 $bucket.weight = 0$ ，选择算法会将此 bucket 对应的桶屏蔽。若所有 bucket 的权重都为 0，此数据包会被丢弃。所以目前 Table-Miss 的处理方式可以由组表转换表达。取消应用扩散算法也比较简单，只要将前 N 个 buckets 的权重设置为 0 即可。或者设置某些 bucket 的权重为 0，即可屏蔽向对应邻居扩散的通路。

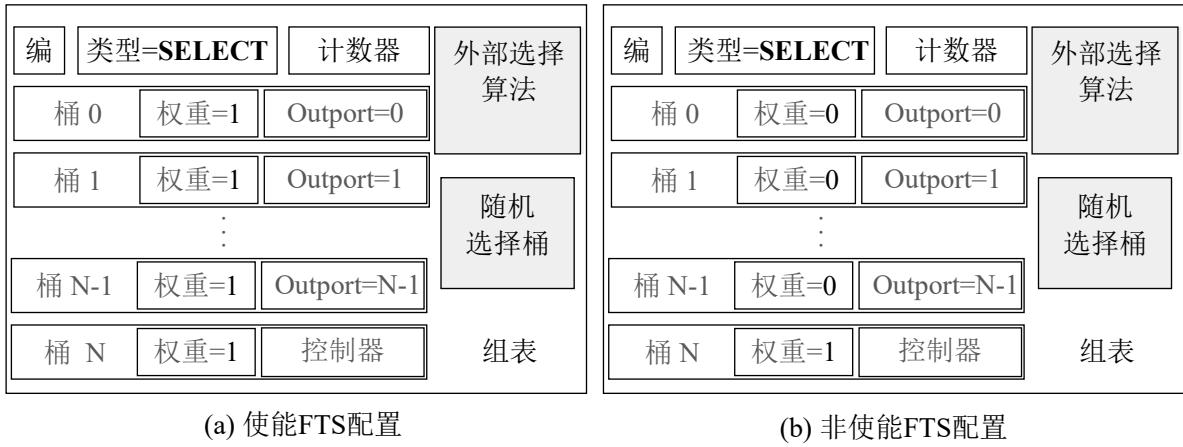


图 2-7 组表的配置方式对比

2.6 系统评估

实验测试包含 2 个部分，分别为：（1）使用软件仿真，对比 FTS 与优化之前转发性能优劣。（2）在真实拓扑中测试流表共享机制的部署开销。

第一部分转发性能测试实验使用了软件仿真平台，由于其仿真性能与硬件转发设备有很大差距，因而本文首先主要针对转发时延，讨论软件仿真实验的系统误差。记：当流表溢出后的流量总时延为 T_{over_flow} ，FTS 优化后的流量时延为 T_{FTS} ，单个数据包转发时延 T_{fwd} ，单个数据包传输时延 T_{trans} ，传输时延统计误差 ΔT_{trans} ，转发时延统计误差 ΔT_{fwd} 。因控制器与交换机都实例化在同一台计算机系统内，因此数据包的传输时延会远小于真实值 T_{real_trans} ：

$$T_{real_trans} = T_{trans} + \Delta T_{trans} \quad (2-8)$$

实验的系统误差 (ζ) 主要来自于转发时延，和传输时延的测量：

$$\zeta_{time} \propto \Delta T_{trans} + \Delta T_{fwd} \quad (2-9)$$

软件交换机通常状况下转发延迟小于 0.2 m s，使用真实硬件测试转发时延则会远小于 0.2 m s，因此：

$$\zeta_{time} T_{fwd} \propto 10^{-1} ms \quad (2-10)$$

一般对于一条远程 TCP 链接数据流的包传输延迟，可估计在 100ms 的数量级，因此：

$$T_{trans} \propto 10^2 ms \quad (2-11)$$

实验总时延 T 由转发时延和传输时延组成:

$$T_{real_trans} \propto T_{fwd} + T_{trans} \quad (2-12)$$

真实的总时延 T 由实验得到 T 与误差组成，并且依据公式2-8至公式2-12可得到真实时延 T 与时延结果 $T(exp)$ 的关系:

$$\begin{aligned} T_{over_flow} &\propto T_{fwd} - \Delta T_{trans} + T_{trans} + \Delta T_{trans} \\ &= T_{overflow}(exp) + 99.9ms \end{aligned} \quad (2-13)$$

$$\begin{aligned} T_{FTS} &\propto T_{fwd} - \Delta T_{fwd} + T_{trans} \\ &= T_{FTS}(exp) - 0.1ms \end{aligned} \quad (2-14)$$

对于公式2-13与2-14，可以得到结论：(1) 引入测量值的误差后，溢出时的总时延高于其他值两个数量级以上，且误差不引起自身测量值数量级（均为 $10^2 ms$ 数量级）的变化。(2) 由于时延误差 $10^{-1} ms$ 远小于 $7.56 ms$ 的测量结果，因此误差可以忽略，因而 FTS 优化后的真实值也达到对其他两个数据的数量级差距。所以证明性能测试实验使用软件仿真是具有代表性的，能保证结果的准确度。此外对于性能测试实验，制作真实硬件交换机造价昂贵，不易配置导致实验手段缺乏灵活性，此番论证后从理论上显著降低了实验的困难程度，易于其他科研工作者进行重复验证。

2.6.1 性能测试：转发时延与控制消息数量

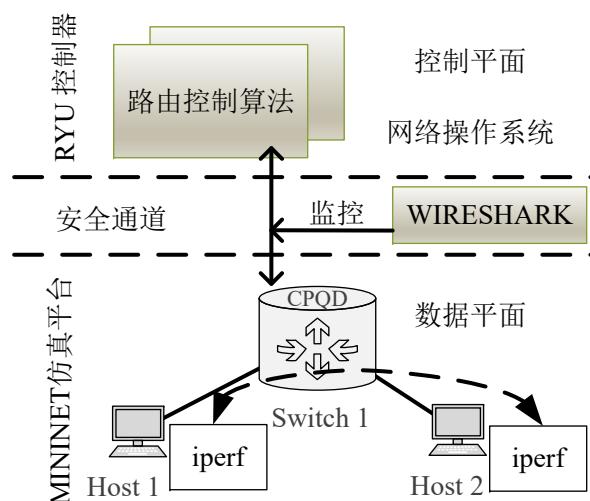


图 2-8 性能测试实验拓扑

本文评估流表溢出对于通信速率和安全通道消息数量的影响，并且将之与使用 FTS 机制后的结果相比较。本实验平台使用 x86 服务器，Intel i3 CPU，8GB 内存，ubuntu14.04，

远端 ryu 控制器。实验拓扑如图2-8所示，性能测试中使用 MININET[22] 仿真工具建立单节点转发场景，使用 iperf 工具分别测试 RTT 和转发速率。利用 wireshark 工具分析安全通道内的控制消息数量。经过多次 iperf 流量测试后可到可信结果，当流表溢出时 UDP 传输有 15% 的丢包率，TCP 传输没有丢包。

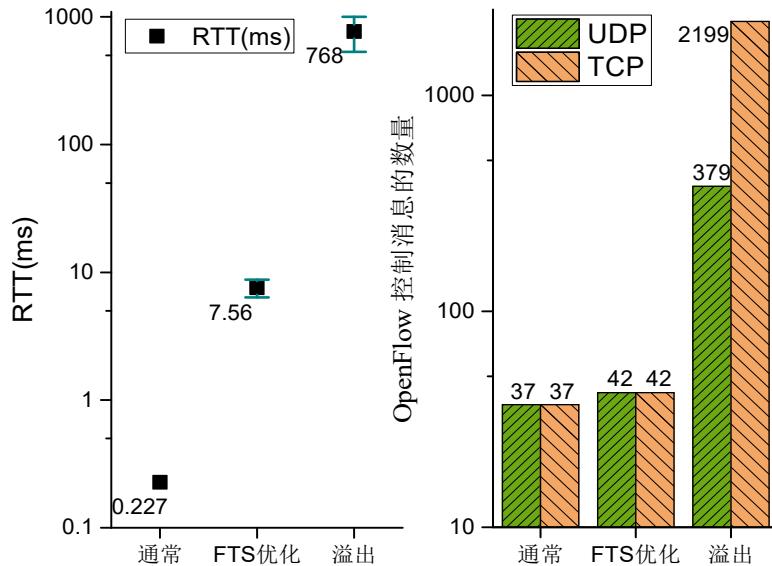


图 2-9 FTS 与普通 OpenFlow 交换机再流表溢出时性能对比

如图2-9的右子图所示，当发生流表溢出时，建立 UDP 与 TCP 传输均消耗大量控制消息，比正常情况高出 1 到 2 个数量级。流表溢出导致既有表项被删除后重新下发原本流表信息，导致控制消息数目大量增加。而 TCP 传输消耗量大，可以通过 TCP 的重传机制合理解释：TCP 协议为保证所有包都到达过目的地，在发生丢包的情况下会重新发送数据包，这部分重传数据包对应着更多的控制信道数据。当交换机采用 FTS 策略时，当流表溢出时控制消息数量比正常情况多出 13.5%，相对最差情况提升了 2 个数量级。

如图2-9的左子图所示，正常通信时数据包的转发平均时延为 0.227ms，流表溢出时转发平均时延增加到 768ms，通信时延比正常情况增加了 4 个数量级。这是因为流表溢出后正常转发的流表项会被删除，导致转发中断，重新安装流表项需要再次建立，耗费时间。因而导致总转发时延增大、吞吐率下降。同时可以看到，当使用 FTS 优化后，发生流表溢出后，转发延迟从 768ms 下降到 8ms 以内，同样优化了 2 个数量级。这是因为 FTS 机制在本地寻找其他转发路径并建立转发，可以大大降低控制器的冗余操作次数，并最多只需要消耗一次流表项的建立时间。

2.6.2 代价评估：额外消耗全局资源与流路径

本小结测试，实施 FTS 机制后所需要的额外的转发资源。在流表容量限制下，建立非最短路径时额外增加的转发距离。

本文选取 CAIDA 公开的自治域真实拓扑^②数据，统计 FTS 机制下数据包到达目的

^② The CAIDA UCSD Macroscopic Skitter Topology Dataset [<http://www.caida.org/tools/measurements/skitter>](http://www.caida.org/tools/measurements/skitter)

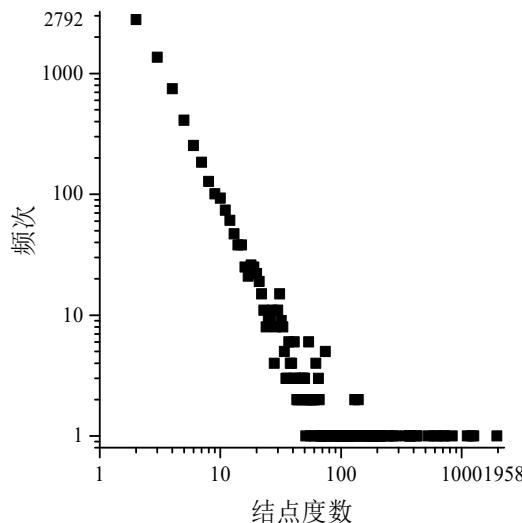


图 2-10 实验用的真实拓扑结点度数特征分布

地的总转发次数。在拓扑内共有 6744 个结点和 26501 条边，结点度数（每个节点的邻居交换机个数）分布可见图2-10。本实验将拓扑中的每个自治域抽象为一个转发节点，可以保留真实拓扑特征，也可以简化实验。

为减小流表溢出现象对交换机带来的副作用，FTS 机制会使用邻居资源作为代价来完成正常转发。由于自主转发的随机性，到达目的地的路径并不一定是最短距离。本文计算了平均路径增加，路径总长度，路径长度与增加值的比例，以及额外总流表资源使用量。收到软件仿真工 MININET 的性能限制，多于数千个节点的仿真很难在其上完成。

第二个实验主要考察路由计算以及统计流表项数量，因而其实无需使用 mininet 完善的网络模拟功能，避免浪费实验的计算性能。本文使用 C 语言来提高仿真性能，实验只观察数据包的转发行为，以及转发节点的拓扑关系，忽略交换机的其他真实特性（例如，转发时延，端口，内部流水线等）。同时也忽略网络多层次结构（如，控制平面，管道，转发平面等），将网络抽象为理想的数学符号，网络功能只保留路由和 FTS 算法机制。

仿真网络的拓扑和功能简化后，还需要打入流量才能组成完整的实验。真实网络中流量巨大，链路数目多，端口连接复杂。本文做实验时无法得到完整的网络流量细节。其次即使得到所有流量状态信息，也不可能在仿真平台上再现真实网络情况。使用真实流量进行网络仿真时，一般需要专用的流量回放设备 [23]，需要专门的软硬件系统，不但造价昂贵，且实验复杂。网络连接中的每个端口都需要同时捕获流量。由于速率大，捕获设备只能保存短暂的场景。在测试时，只能针对单节点网络执行。用此方式完成全网仿真，无异于重新建设全部网络，显然不现实。因此本文计划直接通过逻辑推导得到统计值。

本文的基本思路如下：1) 流量仿真。生成拓扑内每个端口到端口的覆盖所有点对点的流，实现点点之间互联，流表信息是最完整的。2) 溢出仿真，假设仿真针对一个

节点发生流表溢出时的场景。在这个流表溢出的节点，部署 FTS 算法。则可以分别得到向全部邻居节点随机转发后的网络状态结果。将得出的统计值归一化，来表示统计数据的概率密度。此番假设仍有不足之处：本文假设网络中任意两点均可达并通信，但实际情况在同一时刻网络流量并不需要点对点互通。因此实验所表现的场景是网络拥堵时的较差情形。

通过实验本文只能得到变量统计值的平均概率密度，也是网络长时间运行后真实统计值的渐近线。然而面对庞大而复杂的法完成的仿真任务，在实验精度上做取舍依然是值得且有效。

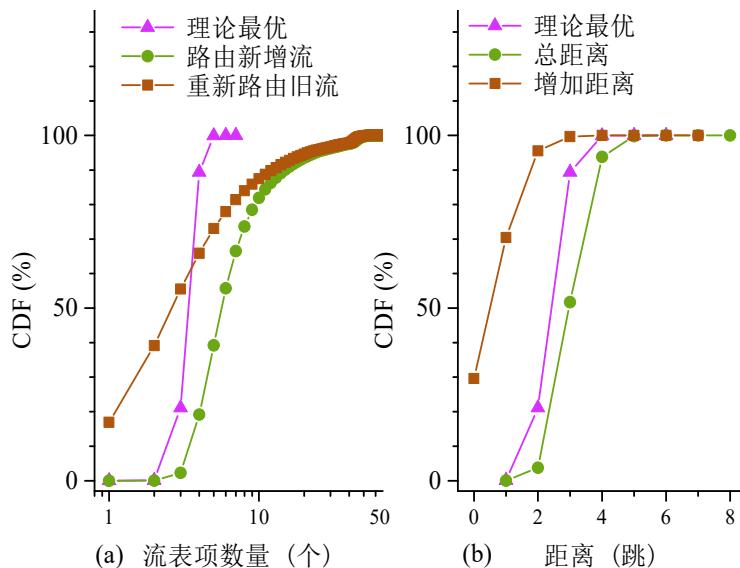


图 2-11 当流表溢出时，FTS 建立一条新流所消耗的额外流表项资源以及每个数据包从源到目的地的转发次数累计概率分布

图2-11 (a) 展示了 FTS 机制建立连接过程中使用的流表项数目的 CDF 曲线。由于 FTS 机制借用邻居交换机的流表项，流表资源总使用率与拓扑内节点平均邻居个数成正比。图中三角线段代表正常情况下流表的使用数量，圆形线段代表对新流实施 FTS 机制所需要流表项的数量，方块线条为正常流突然中断后使用 FTS 机制恢复通信所需的流表项数量。实验计算了激活 FTS 机制之后流表占用量的变化情况。因而方型线条表示某条活动的流被删除后 FTS 机制占用的流表总资源；圆形线条表示新流因流表溢出而无法建立连接的情况。前者平均须多占用流表 5.9 条，后者平均须多占用流表项 8.7 条。正常情况下新增流表项为 3.9 条。可以看出，在一个庞大的广域网中，FTS 机制 90% 概率下新增流表项不超过 15 条。99% 概率下新增流表项不超过 45。最差情况为增加 67 条流表项。

图2-11 (b) 显示了 FTS 对数据包转发次数的影响。三角图线是正常情况下数据包的跳数。圆形图线表示 FTS 机制增加到达路径的平均长度。蓝色图线表示 FTS 机制中，流对比正常情况下多出的转发次数，可以看到 95% 的概率 FTS 机制增加的跳数不会多于 2 跳。

图2-12示出 FTS 处理一条新流时，新流走过的路径的距离增长情况，在 95% 的情

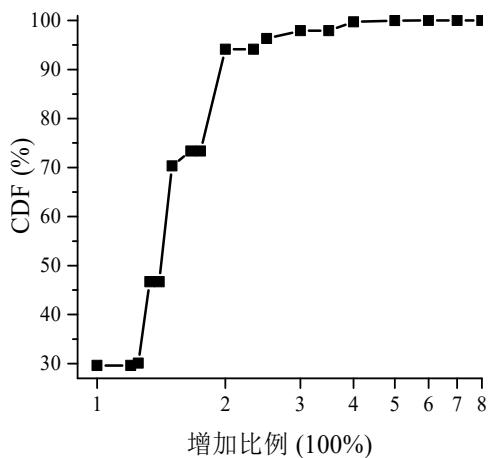


图 2-12 相比最短路径, FTS 完成路由增加的转发节点数量之比

形下路径增长不会达到最短路径距离的 2 倍。本文已将将仿真程序和拓扑数据、实验结果公开到 [github^③](#) 上。

FTS 不能保证同一 TCP 流量均往一邻居随机转发, 并且数据包走过的路径长度也不尽相同。但是从图2-11(a) 的实验结果得到: 包包之间的路径差不超过 7 (8-1=7) 跳。GE[24] 网口标准转发时延为 200us, 因此总时间差不会超过 2ms。这有可能启动 TCP 乱序重排机制, 还有可能增加传输时延。但是, 本文主要期望 FTS 机制能对流表溢出后的性能进行优化, 并与流表溢出后转发设备的性能做对比, 乱序导致的性能下降是可以忽略的。

2.7 本章小结

本章首先介绍了目前 SDN 架构中处理流量的方式, 新兴的 SDN、OpenFlow 交换机是学术界研究的热点。在2.3小节通过流量特征分析流表溢出的危害性以及不可避免性, 相比于传统的网络交换设备, OpenFlow 交换机需要消耗更大的查找表容量, 流表溢出带来的性能下降现象较为严重。本文基于流量分布规律以及理论分析, 在第2.4小节提出基于全局资源的流表共享优化机制, 扩展了 OpenFlow 交换机的 Table-Miss 处理方式。可以有效地缓解流表溢出现象带来的巨大网络通信性能损失。

第2.5小节探讨了方案可执行性已经实现方法, 最后本文在第2.6小节本文评估并证明了本文的方法可有效缓解流表溢出所带来的危害。文中从两方面验证了 FTS 机制, 从实验结果中可以得出 FTS 机制有以下三个优点: (1) 当流表溢出时, FTS 相比于目前的 Table-Miss 处理模式可有效地降低控制消息数量和 RTT 时延达两个数量级。(2) 本文提出了一种有效的快速外部组表选择算法。(3) FTS 易于在真实交换机上部署和控制, 占用硬件资源少, 与现行 Table-Miss 处理模式兼容。

转发设备硬件资源消耗量大以及设备成本高是目前 SDN 网络向广域网推进的重要难点。其中流表资源的匮乏有是该问题的核心点, 本文提出的 FTS 机制将为后续 SDN 发展提供一个可参考的容错机制。

^③ https://github.com/qiaosiyi/test_overflow_random_foward

3 研究可编程硬件设备加速网络交换层方法

3.1 引言

本章利用可编程硬件与交换芯片（ASIC）联合设计的思想，提出了一套异构网络数据平面架构，可同时提供高可编程的灵活性以及可比拟于交换芯片的强大性能。基于软件的数据平面性能低下，这类技术着重于便捷开发，价格低廉，无需在网络中部署专用设备的场景，是快速实现功能的首选方案。目前在虚拟化的云服务系统中，已经部署了大量基于软件的网络功能：（1）转发层，华为 CE1800V^[131] 是专为数据中心云计算虚拟化环境部署的一种分布式虚拟交换机。其支持标准 Open Flow1.3 控制协议，以及 Open vSwitch 数据库管理协议（OVSDB），基于英特尔 DPDK（Data Plane Development Kit）技术提供每核 12Gbps 的转发吞吐，比业界平均水平高出 20%。（2）流量监管，Activelogic^[132] 是一个提供安全可靠、流量分类、提高 QoE（Quality of Experience）能力的网络管理工具。它基于软件可自动化部署，依靠超大规模、人工智能技术以及云计算场景优化的能力，在数据平面解决流量监管的问题。基于软件的数据平面功能可以依靠堆叠 CPU 核数来实现大规模的性能扩展，但由于计算复杂度过高、基于指令的图灵机在高速内存共享和海量数据处理场景中效率低下，即使简单转发的性能达到 100Gbps 线速也需要占用 10 个 CPU 核心^[17, 133]。综上所述，单纯地依靠软件处理器扩张来增加网络性能边界收益将越来越小。

本文提出一种高性能的数据平面可编程硬件架构，自适应交换结构（Adaptable Switch, AS）。通过 FPGA 与交换芯片联合设计的思想，AS 架构可同时提供 FPGA 的高灵活性与交换芯片的强大性能。论文在前述硬件设计模型的基础上，继续研究硬件逻辑高度并行的性能大规模扩展方法。论文设计了 ASIC 面向可编程硬件的扩展接口。交换芯片将数据包头拆分并通过高速数据互联载体发送给 FPGA，利用 FPGA 可重配特性实现完全可编程的包头处理。AS 架构解决了 FPGA 性能差与资源少的限制、增强了网络芯片的可编程能力，并且还提出了一套部署在硬件上的高资源利用率的并行流水线和流表分配优化算法。在之后小节中陆续解答本章研究内容下的三个问题：如何设计一种同时兼顾转发性能和可编程能力的交换设备？如何利用现有设备优势，对其做最小改动以满足设计？如何实现高资源利用率、高灵活性的高性能硬件可编程数据平面设计方法？

3.2 问题背景概述

虽然硬件交换机在网络性能方面大幅超越基于通用软件服务器的数据平面^[18, 133]，但是基于白盒交换机和数据平面可编程专用芯片的编程能力受限。符合 OpenFlow 规范的白盒交换机可将控制平面移交给远端软件层，从而实现设备的再开发能力。在 DDoS 防护、负载均衡等基础网络转发设备的智能化和可定制化方面给出了比较好的灵活性。阿里巴巴在其云计算网络场景中，通过可编程硬件交换机和通用服务器结合来实现公

有云的网关服务。此架构既拥有芯片带来的网络转发性能提高 (6.4Tbps, 400ns 延迟) 也具有可编程带来的网络功能快速部署迭代能力, 还能实现软件所擅长的复杂网络调度功能^[134]。这样同时兼顾了性能、灵活性, 在大规模扩展网络体系结构时达到降低成本, 满足业务需求和简化网络架构同时提升服务稳定性。数据平面可编程芯片提供了硬件层面上的可编程包头抽取器、可编程流表以及可编程执行器。其依靠快速查表 (TCAM, SRAM) 法, 或经过后期编程选取特定的冗余逻辑模块 (在 ASIC 芯片内部的空间上堆叠的可编程单元) 法, 来完成专用电路 (ASIC) 的直接逻辑描述^[19, 59]。不过这类可编程芯片架构提供的可编程性也不是完备的, 可编程流表限制了查找的宽度、深度范围, 造成了逻辑资源浪费以及流水线处理时延较大。同时, ASIC 设计定型之后无法增加新的用户特性 (状态转发、随路计算、监测计数和包调度特性), 导致这类 P4 专用芯片的可编程范围是大大受限的。

软件定义网络将网络的可编程性抽象为对流表的灵活配置操作, 此外数据包协议无关处理器 (P4) 使得数据包匹配域可自由定制, 更加扩展了可编程数据平面的编程灵活性。P4 处理器作为目前网络编程领域标杆技术, 其有能力实现自定义包头解析, 自定义流表的组成结构, 以及很高的吞吐设计容量 (一般可达到 10Tbps), 在科研与市场环境下已经得到了良好的应用反馈^[21]。

但在真实应用环境下, 有很多定制化应用无法在高速的数据通路中部署, 如表3-1所示, 只能将其转移至服务器软件端, 或者在 FPGA 的开发验证平台中验证, 这无疑使得面临巨大的性能损失。虽然 P4 网络芯片极大地解决了 OpenFlow 编程能力不足以及其设计本身所带来的可扩展性差的难题, 但它不具备真正意义上的“软件”可编程性。

表 3-1 不同类型可编程数据平面对比

平台	吞吐性能	可编程转发	事件触发 ^[135]	状态转发 ^[136]	函数计算 ^[105]
软件平台 ^[17]	10Gbps	✓	✓	✓	✓
FPGA 平台 ^[65]	100Gbps	✓	✓	✓	✓
P4-芯片 ^[59]	12Tbps	✓	✗	✗	✗
设计目标	>1Tbps	✓	✓	✓	✓

根据相关工作简介第??小节介绍以及在实际场景下的各类需求, 我们总结出 P4 处理器概念上的一些不足之处, 并总结为以下三点: 1) P4 数据通路内的处理动作只能被数据包触发, 而无法响应其他行为。针对一些 QoS 场景系统更希望根据队列深度做出一些响应, 比如拥塞控制场景下的控制算法 NDP^[135]。2) P4 缺乏计算、控制属性的指令集, 例如乘除法、灵活分支判断等, 例如网络随路计算^[92]、网络测量^[102, 105]等领域。3) P4 为无状态转发的可编程流水线设计, 则对状态协议处理以及有状态计算造成了困难 (防火墙^[137]等)。

为应对以上缺点, 本章设计并提出了一种具有支持适应性计算能力的可编程数据平面的自适应交换系统 (Adaptable Switch, AS), 第3.3小节介绍了系统设计目标, 以

及系统提供的开发框架使用方法。第3.4小节介绍 AS 架构设计核心思想：自适应交换系统同时利用了可编程硬件设备的高度灵活性，以及 ASIC 交换芯片的高性能特性。本文使用 FPGA 作为可编程硬件的具体实现载体，FPGA 具有与硬件类似的数据包处理能力，同时具有高度灵活的二次编程能力，可带来解决 P4 架构灵活性不足的应对方案。同时本文修改了通用的基于 ASIC 的交换芯片的结构，增加与高度灵活的可编程设备通信接口，为后期灵活开发做基础，并尽量减小 ASIC 交换芯片的修改量，以减少设备厂商的研发改进费用。第3.5小节对优化 FPGA 内硬件性能以及效率做出了分析，并将该优化问题归约为一种 NP 难问题，此问题由3.6小节中给出的一种启发式算法解决，最后3.7小节评估了 AS 系统的资源消耗以及性能，并在其中实现了 P4 可编程网络架构无法实现的拥塞控制、网络测量、有状态防火墙的三种应用。

3.3 系统架构介绍

P4 是目前广泛使用的可定义数据平面内的数据包解析和查表过程的语言。P4 将流水线外的功能需求通过 P4_extern 模块来让用户自行定义。然而 P4_extern 的编程范围只在 PISA 提供的执行器序列之内，例如校验算法，位移，插入数据等。目前的 PISA 芯片架构对于“PISA_extern”的功能需求没有支撑能力。一种显而易见的解决方案是当需要一种新的“PISA_extern”时，重新设计一种支撑新特性执行器的 ASIC 硬件。但这与 PISA 高灵活性的设计初衷相矛盾，从经济实用性角度分析也是不合理的。

本文提出一种硬件异构型的交换机架构，可以支持任意 P4_extern 所定义的功能。本文将这种结构命名为“自适应交换”，首先本文需要解决如何设计硬件结构使得灵活性和性能可以充分展现，其次本文需要解决如何开发编程语言以映射到自适应交换的数据平面。最终为了展示自适应交换的高扩展性，本文将上小节内提出的 PISA 难以应用的场景均在自适应交换平台中实现。

3.3.1 自适应交换架构设计

自适应交换的硬件架构框图。自适应交换包括两部分，1) 固定功能的 ASIC 交换系统（Switching System, SS）；2) 用户定义可编程逻辑块（Programmable Logic, PL）。SS 基于标准的交换芯片（switching ASIC）处理功能，PL 利用 FPGA 类的可编程硬件支持用户自定义逻辑。一个标准的交换芯片功能包含了数据包头的解析，基本的 ACL 功能和各个字段的匹配以及执行转发/丢包等动作。额外还会包括流量管理、队列调度等 QoS 功能。架构中 SS 与 PL 两部分可由双芯片拼接的方式实现，SS 部分利用一个标准传统的交换芯片（既可以支持 P4 编程，也可以不支持 P4 编程）构成。PL 端使用 FPGA 芯片或由 FPGA 构成的多用途片上系统（MPSoC/ACAP）芯片。在双芯片拼接方案中，两个独立的芯片系统须通过高速互联总线相连接，例如 PCIe 接口，以太网接口或者类似的收发器。相应地自适应交换系统也可以由单独一颗芯片构建而成，PL 与 SS 部分通过片上高速总线相接，例如，AXI 协议总线。

如图3-1左半部分所示，交换网络是 SS 的核心组成部分。交换网络支持网络入端

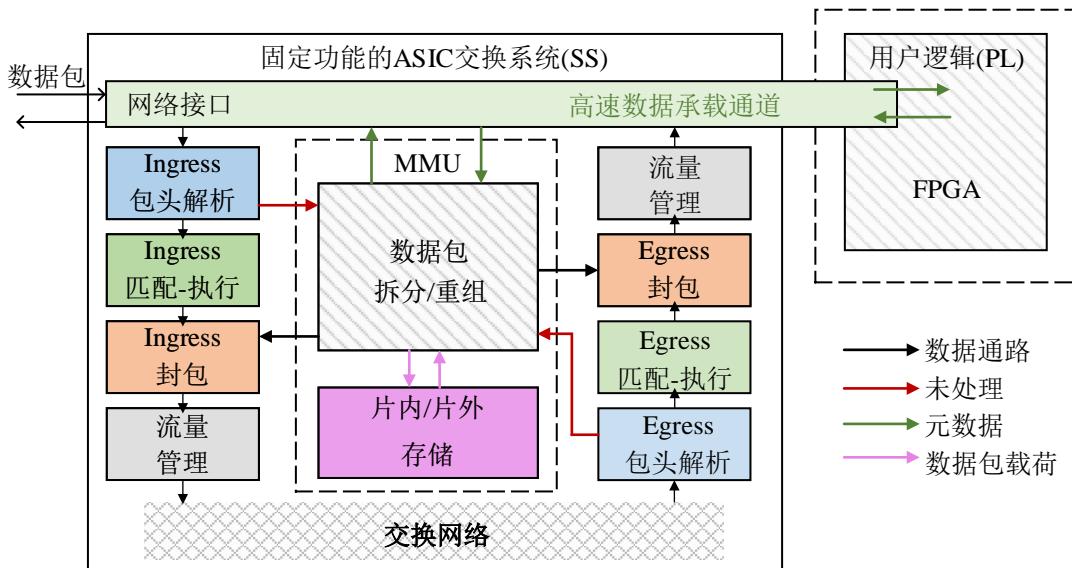


图 3-1 自适应交换结构框图

接口与出端接口多对多的数据包互联传送，一般由交叉开关（cross-bar）构成。数据包经由物理网络接口进入 SS 系统，在交换网络两边，数据包会经过入端（Ingress）处理流水线，以及出端（Egress）处理流水线。在两条流水线的组成分别有包头解析（抽取必要的包头域数据），流表（对包头域数据进行匹配和执行相应的动作集），封包（重组/修改数据包结构），以及流量管理（包缓存/调度/限速等）。

数据包进入自适应系统时，首先经由 SS 端的处理，通常情况下大部分数据包可以完整地被 SS 处理，并返回外部网络。只有在 SS 内的功能无法满足需求的那部分数据包会再次送入 PL 端做协同处理。对于送往 PL 端处理的数据包，在 SS 端在片上存储中（或使用片外存储）也保留有一份完整的数据包备份，SS 只将数据包的元祖数据信息送给 PL。元祖数据中包括了 PL 完成处理所需要的定制化的包头信息或者其他描述信息。PL 处理完成元祖数据后，会更新重写元祖信息中的包头以及描述字段，再返回给 SS。最后 SS 将备份的原始数据包与更新的元祖数据重新整合成一个完整的数据包后再次对其进行转发操作，或简单丢包。

本文将在传统的交换芯片中增加一个包存储管理机制（Memory Management Unit, MMU），图3-1中左边虚线内部所示。当数据包元祖信息送往 PL 处理时，这个数据包在 SS 端的完整备份由 MMU 管理。MMU 应包含三个主要功能：（1）动态申请/释放数据包的存储块位置及其大小；（2）驱动数据包向内存中读写时序；（3）读出备份数据包后与从 PL 返回的新元数据对接重组。

上述自适应交换流程与原理是建立在本文对以下两方面的分析结果和假设之上。首先，在 PL 中处理的信息通常只依赖于包头，或者数据包包首部分的字段。在设计中被交换到 PL 的元数据可以被灵活地定义，并且保证只占用 SS 与 PL 之间一定量的通信带宽成本。在某些极端的例子中，如果处理过程需要数据包所有的数据时，元祖数据也将包含一个完整的数据包内容。第二，不是所有的数据包处理都需求调用 PL 端的

功能。否则，如果有一种功能是普遍适用的，则一定会集成在现行的通用交换芯片内。或者这种功能需求可以直接拆分出来一部分由 SS 端完成处理。

自适应交换的高性能来自于对网络通用数据包长的分析。本文考虑到，目前网络内平均包长度为 600 字节左右，假设元祖数据包含整个包头部分（64 字节之内），则 SS 与 PL 之间的通信成本只占需求流量的 10%。即，利用单插槽的 PCIe4.0 接口（256Gbps）作为高速互联，自适应交换系统可以将目前最高速度（12.8Tbps）交换芯片中 20% 的数据（>2Tbps）卸载到用户硬件可编程逻辑中进行处理。这远远超过了（ $\times 10$ 倍）单纯由 FPGA 组成的可编程数据平面的处理性能，也是本文研究的最主要的动机。

3.3.2 用户定义功能的开发流程

对于基于 FPGA 的 PL 端，基本的开发设计流程包括如下几部分：

- 定义数据包处理需求以及所需要的数据流模型。
- 编写处理规则对应的函数或流程代码。程序代码可包括高级语言如 P4/P4_extern、P4_FPGA，或者底层的硬件描述语言如 verilog HDL。此外包括数据平面高层次生成系统，如 SDNet^[66]，也可以用于加速 PL 端的开发工作。
- PL 端的机器码编译器。不同 PL 目标器件下的编程，往往有不同的编译步骤。对于基于 FPGA 的编译器，本文使用 FPGA 芯片厂商提供的编译环境。但本文最主要的贡献是解决如何将数据包流处理模型在异构形态下重新组织，并且完成高性能的异构形态下包处理逻辑无语义偏差的拆分和翻译。

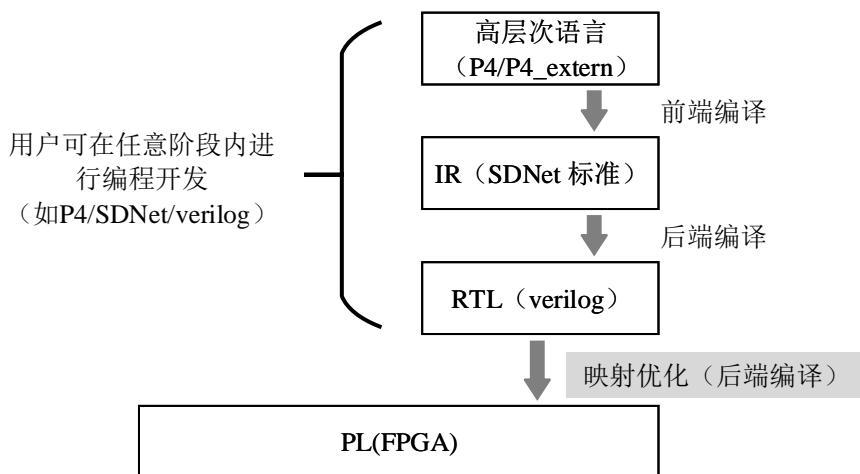


图 3-2 基于 SDNet 工具链的 FPGA-PL 端编译流程

本文利用 Xilinx SDNet/P4-SDNet 作为建立本文设计原型系统的基本开发工具，基于 FPGA 的 PL 编译流程如图3-2所示。SDNet 和 P4-SDNet 已经是成熟商用包装解决方案，并包含了为 FPGA 数据平面设计的 P4 语言到 verilog 硬件模块的编译工具链。P4-SDNet 内建两种专为 P4₁₆（提取包头数据，修改包头域值）提供的 P4_extern 功能。这种 extern 功能给用户提供了足够灵活的直接修改原始数据包头域的能力，但目前此编译器并不支持其他 extern 对象的编译，且用户也不可定义其自己的 extern 对象。

使用其类似的思路，本文在开发流程中增加新的编译模块，使其将能够使用户扩

扩展 `extern` 对象定义的范围：扩展前端编译器，使其支持其他的高层次描述（描述中增加标记符以提高后端编译器的性能），并将其转译为逻辑中间表示层（IR）；最后通过后端编译器再次映射到 PL 端。本文提出的硬件转发体系结构与通行产品不同，下面将详细介绍本文开发流程与经典可编程数据平面（以 P4 为例）之间的差别。

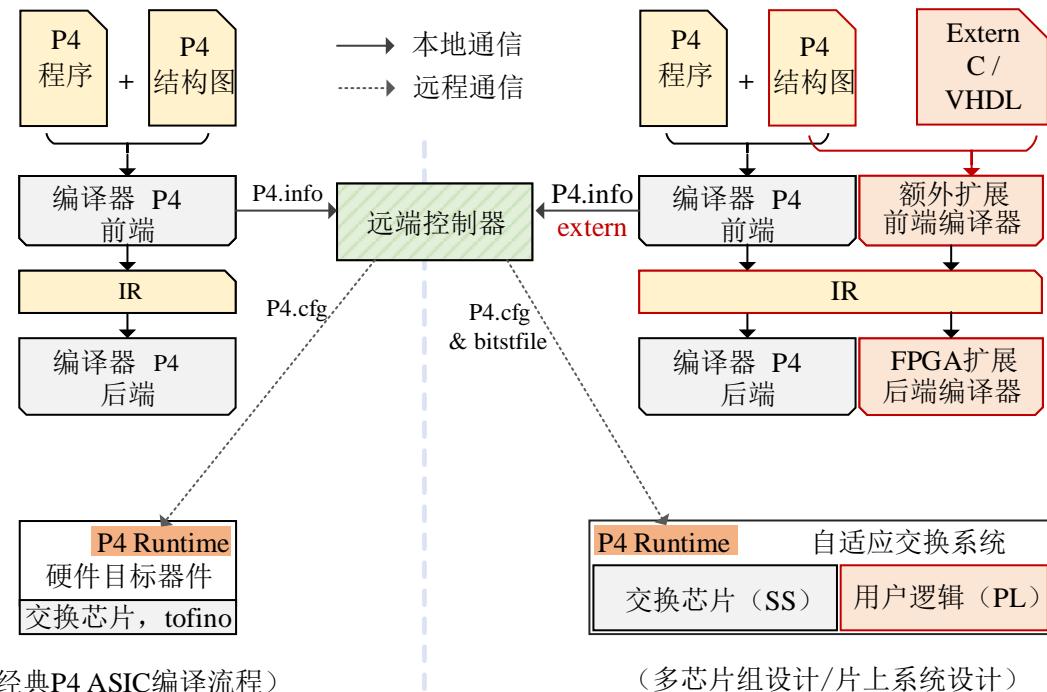


图 3-3 基于 SDNet 工具链的 FPGA-PL 端编译流程

图3-3中左边为经典的P4程序编译标准流程，对P4目标器件编程需要两个描述文件。首先是P4行为描述程序代码，包含了待处理数据包的包头域定义，包头域匹配关系图，流表设计，执行动作设计等。第二为P4架构图，此文件由设备厂商提供，编译器须根据不同底层芯片实现样式进行有针对性的编译，这样编译器可根据结构图对程序进行等效变换以及重组。前端编译器会将文本代码转换为逻辑关系表，也称作中间表示层（Intermediate Representation, IR），后端编译器一般有厂商提供，可将符合规范的IR表示直接转为机器二级制配置文件。远端控制器将配置文件下载入可编程交换机设备即完成了数据平面重配置过程。此时，控制器也需要收集P4定义信息，以方便后续添加流表项等网络功能设置。

根据之前本文提到的自适应交换系统的额外特点，如图3-3右边所示，本文对基础编译框架进行补充。首先 P4 结构图需要修改为自适应交换机的底层实现，例如需要体现包头域协同处理机制。其次本文支持高层次语言的 PISA_extern 对象编程，因此在额外扩展的前端编译器中，需要将高层次编译器适配入本流程。新增部分由红色区域块所示。最终后端编译器配合 P4 结构图生成 FPGA 中的 RTL 代码，交由 FPGA 厂商工具链生成二进制流文件（bitfile），通过板上运行时代理下载入 FPGA 即完成了 PL 端配置。SS 端配置同经典过程所述。

3.3.3 高层次语言映射样例

代码 3-1 数据平面接口：PL 端高层语言描述

```

1  extern extern_example{           // 声明模块名称: extern_example
2      ext_type(in bit<wdth> input_Port); // 声明接口位宽以及数据流方向
3      int a = VHDL_method_example();    // 模块使用其他语言快verilog
4      void CPP_HLS_method_example(a); // 模块使用类C的函数描述语言
5  }
6  control control_example{         // 实例化样例模块
7      extern_example(0x0) my_extern_example;
8      action my_extern_call(){        // 调用PL端功能
9          my_extern_example.C_HLS_method_example();
10     }
11 }
```

PL 以 FPGA 芯片为例，自适应交换平台可以将 SS 交换芯片无法完成的用户定义功能映射到 FPGA 中。用户的额外功能对象须遵循 PL 端输入输出接口规范。P4 语言风格的数据平面接口规范如代码 3-1 所示。代码定义了一个名为“extern_example”的 PL 模块，模块可由 C 语言或 HDL 风格实现。对外接口位宽以及方向需要标记。在后期可实例化一个或多个样例，每个样例可自由调用 PL 端模块中的任意描述功能。

代码 3-2 extern 实例的 C++ 类声明

```

1 #include <hls/hls_sim/extern.h>           // 头文件extern.h 定义了外围数据传输
2 using namespace std;
3 template <typename .. Args>
4 using ActionPrimitive =
5     hls :: ActionPrimitive <Args ...>;
6 using hls :: Data;
7 class extern_example: public ExternType{
8     public:
9         void init () override {}
10        void C_HLS_method_example(){...}
11    };
12 HLS_REGISTER_EXTERN(extern_example);
13 int import_extern_example(){return 0;}
```

如代码 3-2 展示了 C++ 风格 extern 函数。头文件“extern.h”中定义了核心函数之前的流水线结构（需要用户注意并修改），无需提现在核心描述代码中。FPGA 高层次编译工具链（SDx, Bluespec System Verilog）可读取此类文件，并综合成带有用户功能的 IP 核，小的 IP 核可便于集成进更大的 FPGA 工程。

代码 3-3 extern 实例的 VHDL 模块声明

```

1 'timescale 1ns/1ps
2 module VHDL_method_example#
3     parameter DATA_WIDTH = D_WIDTH,
4     parameter CTRL_WIDTH = C_WIDTH,
5     parameter EXTERN_REG_WIDTH = R_WIDTH,
```

```

6      ...
7      )( //----data interface
8          input      [DATA_WIDTH - 1:0] in_data,
9          //----register interface
10         input [EXTERN_REG_WIDTH - 1:0] in_reg,
11         //----misc
12         input                  clk,
13         input                  reset );
14
15     // local parameter
16     // wires/regs
17     // modules
18     // method example logic
19   endmodule //VHDL_method_example

```

代码 3-3 展示了 VHDL 语言实现 PL 实例模块，在 VHDL 中须描述数据总线位宽，寄存器通路接口等。对于其他硬件系统，用户只需要指定通信用同步数据总线的时钟周期（时间约束）。实际上，本文所描述的所有编译工具链仍处于设计阶段，并需要大量代码工作完成自动化处理。本文主要目的在于规划框架以及作为商用工具链的补充设计参考。硬件开发时间周期过长，为快速进行数据平面验证，本文生成的所有 bit 文件均为手动整合代码。

3.4 自适应交换架构的硬件设计

3.4.1 数据平面内协议设计

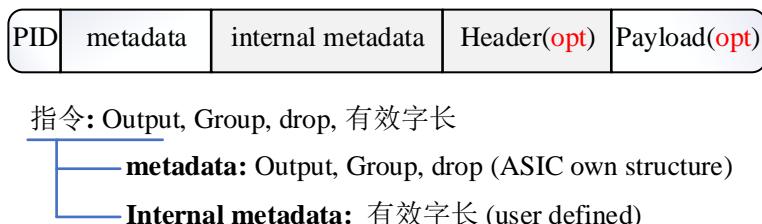


图 3-4 SS 与 PL 内部通信帧格式

上节提到自适应交换架构中 SS 端与 PL 端的通信须包含元祖数据以及包头，本节论文详细讨论通信帧格式。本文设计了一种通信帧（可满足大部分需求，用户也可以自行定义），格式如图3-4所示。首先帧头指定了帧的编号 PID，帧由 SS 中的数据包生成，完整的数据包与帧使用同一个 PID 编号，方便 PL 协处理器处理之后重新组合。元祖信息（metadata）包括了数据包转发过程中的信息，例如输出端口，组表编号，丢包等。用户元组信息为帧有效字长，由于用户可以定义帧长度，这将给用户极大的灵活性空间。此外用户还可定义包头长度以及携带的数据包负载信息长度等。SS 与 PL 之间传递的大部分操作数都是商用 ASIC 中已经实现的指令，因此 SS 端设计可以极大程度复用已有芯片设计。须增加的功能只有帧长度信息等域，这样可使对 ASIC 改动降到最低。

3.4.2 SS 端固定功能

SS 端对于目前商用普通交换芯片中需增加的功能为第一小节中提到的图 3-1 内左边虚线内的 MMU 模块。自适应交换的核心思想是利用基于 ASIC 的交换系统 (SS) 提供包交换的高性能, 而用可编程硬件协处理器 (PL) 提供包处理的灵活性。因此在 SS 端本文将包处理逻辑做的最简化, 只保留一个用户可配置逻辑 (即帧长度的定义)。MMU 将用户定义的不超过帧长度的包头数据 (指某一区间段) 复制到通信帧中即可。MMU 的其余工作还包括了动态申请和释放数据包缓存空间, 以及重组数据包。接下来本小结主要介绍 MMU 的分析与设计。

假设 SS 与 PL 之间单向通信带宽有 256G(PCIe4.0), 以帧内包含完整 64 字包头节为例, 则包速率达到 380Mpps。目前交换芯片的时钟驱动频率大约在 1GHz 到 1.5GHz 之间。对于 MMU 模块, 若留给每个数据包的处理时间 3 周期则不会超过时钟频率 (需要约 1.2GHz 主频)。在之前的 MMU 设计中, 内存被分割为每页 256 字节。调度器以页为单位进行分配, 维护逻辑映射与物理映射之间的关系由于页容量比较小而变得比较复杂。因此 MMU 瓶颈处理一个分配任务需要耗时 25 周期^[65], 尽管这种设计思路在资源利用率上比较占优 (平均资源浪费仅为 1/2page), 但它的时间复杂度太高, 导致分配性能受影响。在 ASIC 的存储设计中, 其经济性要好于由 FPGA 组成的片上存储资源。因而本文在设计存储单位时将每个地址对应的容量配额扩大为 1.6KB (可完整放下一个最大包)。在分配以及查找长包时, 可节省逻辑地址与物理地址之间多次正向/反向映射过程, 因此大大提升分配速度。

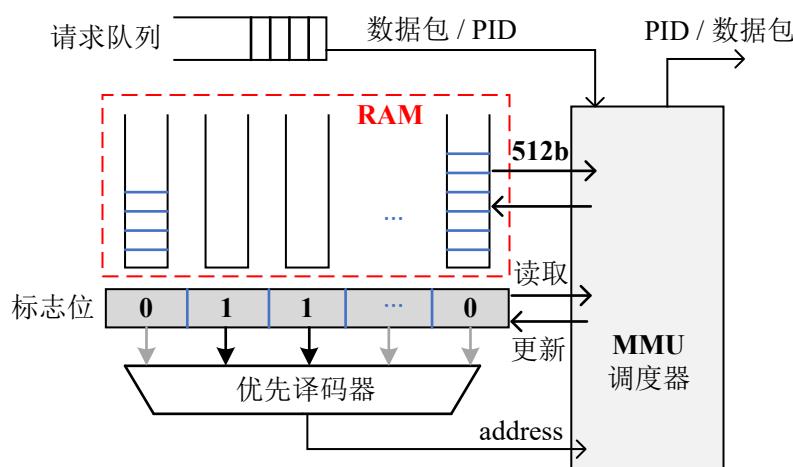


图 3-5 高 pps 性能 MMU 的实现方案

如图3-5所示, 本文使用了一组“标志位”寄存器来指示其对应的 page 空间是否被占用。当数据包请求一个新的可用地址时, 优先译码器会将标志位最左边为“1”的位置作为地址 (address) 返回给调度器, 且便立即将此位置的标志“1”更新为“0”。之后调度器会把数据包向此位置写入, 并且令此位置的地址作为 PID 值填入 PL 通信帧的首部。当 MMU 收到一个含有 PID 值的读请求时, MMU 会读取以 PID 值为位置的存储空间并将其组合为数据包重新发出到入队流水线中, 完成后立即将此位置的标志位为

“1”。此时数据包的 metadata 中一般包含了由 PL 端处理完成给出的执行需求，SS 端无需对此包做额外的分析工作。为了防止地址冲突以及标记位数据不同步带来的读写错误，优先译码器必须在完成上一个“读/写”请求之前，更新空闲地址，以备调度器无缝衔接下一个写请求，读请求则无需使用优先译码器结果，无操作风险。上文分析过，对每个读写操作，MMU 都预留了 3 个时钟周期的处理时间。在系统运行后，由于数据包连续到达以及连续读写，因此在一段时间内，平均读次数是等于写次数的。MMU 将读写交替进行，可为写请求争取 6 个时钟周期的译码时间，一般来讲对于有 4k 个位置的优先译码器，6 个时钟周期足够通过硬件并行地两两比较找出最优位置（每周期比较两次的情况下 $2^{6 \times 2} = 4096$ ，但一般可以比较次数更多）。

接下来分析 MMU 应维护的内存容量大小。在元祖信息帧由 PL 返回之前，数据包都需要维持在 MMU 的存储空间内。假设信息帧处理时间为 $T = n$ (微秒)，上文提到总体包处理频率不超过 $v = 380\text{Mpps}$ ，因而平均在等待的数据包个数为 $T \times v \leq 380n$ 。目前板间硬件处理时延应保证在 3 微秒以下，如果使用片内 AXI 总线互联，则完整处理时间应保证在 1 微秒^[138] 以内^①。为保险起见，本文将处理时间余量增大十倍假设最大不超过 10 微秒，则根据上式，可约估 SS 端等待数据包个数为 4k 个。上文提到每个数据包存储位置占用 1.6KB，则板上共计须设置最多为 6.4MB 存储空间。以目前的半导体工业水平可满足此需求。

3.4.3 PL 设计

基于 FPGA 的 PL 设计最大的挑战来自于大批量信息帧的处理速度 (380Mpps)。对于 PL 内部逻辑，由于帧内包含了基本的 metadata 信息以及包头域段落，对 PL 的性能设计可将其按照网络处理最小包 (64 字节) 的方式分析。按照 FPGA 主频 200MHz 计算，即使每个时钟周期都能由流水线处理完成一个数据包，FPGA 的性能也只能达到 200Mpps。对于某些带状态阻塞性处理机制 (如有状态查找表)，平均需要 2-3 个时钟周期才可处理完成一个数据包，因此 FPGA 的处理性能会下降到 60Mpps。远远达不到系统设计的需求。

一种思路是将 FPGA 内一条流水线完整复制多份，以均分总体任务量。例如可以将流水线同时复制 K 份，平均总处理性能是每条子流水线的 K 倍。然而这也意味着总资源提供量需要耗费之前每条流水线的 K 倍。但由于数据包流水线内有查表等大资源消耗量组件 (1k 深度的 TCAM 表项可占据整个 FPGA 逻辑资源的 1/3)，有限的 FPGA 逻辑面积不可能满足多条并行流水线的资源占用需求。

为解决上述挑战，本文提出一种微结构并行数据包处理流水线设计 (micro-level parallel processing)。微结构流水线的核心思想是将原有的大表项拆分为多个小表项的

^① 在数据包的处理过程中查找 TCAM 表是最费时间的，一般需要 20 个时钟周期，但正常处理流程内 TCAM 表项不会超过 2 个。普通精确匹配查找表只需要不超过 3 个时钟周期即可。我们假设比较悲观的情形，PL 内的处理过程包含 5 级 TCAM 查表以及 10 级精确匹配查表，则总共须耗费 650 时钟周期。若以 200MHz 主频运行 FPGA，则总耗时约 0.7 微秒 (一般只需要 0.1 微秒)。因此本文认为，PL 的处理时延主要由信息帧传输过程贡献。PCIe 收发器结构复杂，突发的批次传输时延较大 (100 时钟周期^[138])，但高性能片上 AXI 总线的传输时延也很低 (不超过 10 时钟周期)。

等价组合，即通过不完整的复制大资源模块，而满足原始用户语义下的数据包处理需求。总结如下，本文将通过以下两个关键技术解决高效 PL 处理设计，1) 提出一种资源高效的并行数据包处理架构，2) 通过流表编译算法将流表项映射到多路子流表中，以达到流量均衡并高效利用硬件资源。

1) 资源与性能

查找表是数据包处理过程中最核心的部件，也是资源消耗最大的组件^[139]。为提升效率本文需要依靠并行子流水线的思路扩展数据包处理性能。同时，为满足资源消耗限制的约束，本文需要分析如何拆分流表。另外，多级流表是目前交换系统中流水线的重要设计特征。多级流表可以降低单张表的资源消耗，但每条子流水线中只包含部分信息，无法保证同一条流水线内数据包的完整处理诉求。

2) 并行流水

流水线被分为 K 个并行子流水线后，数据包流也应被尽量平均地分类为 K 份。然后将不同的组份以流量均衡的形式放置在不同的子流水线中。本文将数据包按照包头信息内的 ID 比特位来分成 $j (j > K)$ 个组份。例如，将包头信息哈希为 n bits 数值，以此数值来代表不同的流组份（根据哈希函数定义，同一条流的所有数据包均可分到同一个组份），则共可分成 $j = 2^n$ 份，显然每一份的流量大小是不一定相等的。如图3-6所示，本文将 j 个 ID 组份以流量均衡的形式安置在 K 条不同的流水线中。

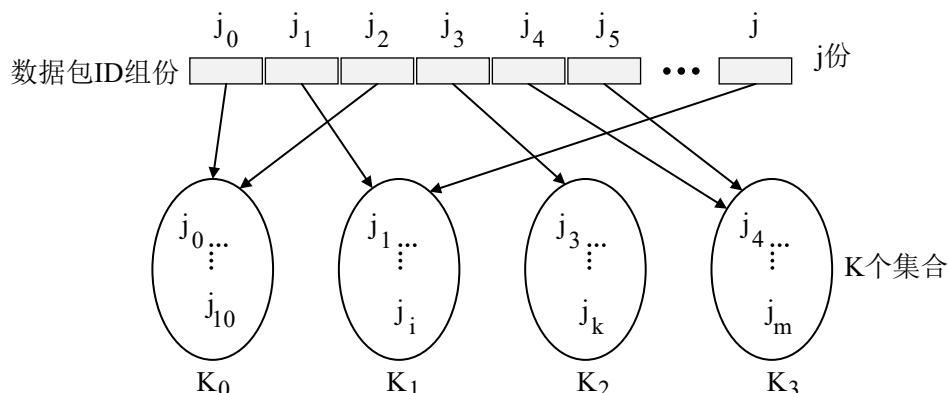


图 3-6 j 组 ID 组份向 K 个集合中分配

为确保每个 K 集合中流量均衡，可调整 j_i 的排列，若假设 j_i 为第 i 个组份的流量大小，使得每个集合之间的总流量大小相近。则集合总流量 K 可表示为：

$$K_0 = \sum_{i_0}^{k_0} j_{i_0} \quad (3-1)$$

3) 流表拆分

本文首先讨论在 PL 端不同类型的流表拆分原则，然后讨论如何使拆分表组合为多级流表之后的语义保持不变。查找表从功能以及实现方式可以分为精确匹配和掩码匹配。由绪论部分可知，精确匹配目前有两种实现方式：(1) 内容地址映射表 (CAM)，(2)

基于 RAM 的查找表；掩码匹配可由 TCAM 实现。

前文提到，数据包在进入子流水线之前须经过哈希函数（hash()）分类，并确定最终去往的目标子流水线。将流水线分为 K 份，则流表也需要拆分为 K 份。哈希函数将全体数据包流分为了 j 份，因此利用哈希函数也可以将流表内容分为 j 份，并按照集合 K 中的分布，将流组份 j 对应的所有流表项也分配到 K 个流水线中。下面依次介绍每种表的拆分方法：

- CAM 表。此过程对于精确匹配的 CAM 表的分类比较直观，只需要按照哈希结果，将 CAM 表中每一条流表项分到对应的 K 集合内，亦可保证语义正确。
- LPM (Longest Prefix Matching, 最长前缀匹配) 表。与 CAM 不同，PLM 表内含有掩码位 (*)。不同的数据流被哈希函数分到不同的组份，由于不同的组份有可能被分到不同的子流水线，有可能导致被同一条 LPM 表项同时匹配的两条流被分到不同的子流水线。若按照无复制方式去拆分流表项，则必然造成一部分流量无法匹配成功。如图3-7所示，在做流量分类时，选取包头域中四个 bit 位进行哈希取值。若这些 bit 位中有位置对应到了带掩码表项 ($1*10$)，而此时，两条流“1010”与“1110”恰好又被哈希函数分到了两个子流水线中的组份 (j_1 与 j_2)，则此掩码表项须分别复制到两条子流水线中。

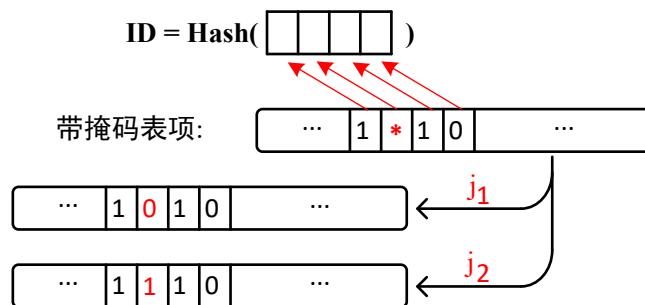


图 3-7 带掩码表项在拆分时，可能需要被复制多份

因此带有掩码的表项在做并行子流水线拆分时会占用比拆分前更多的表项空间。例如在 IP 路由查找场景下，通过分析掩码的位置以及流组成发现^[139]，适当调整包头域中抽取的 bit 位作为哈希函数的输入值，可保证多占用的表项空间维持在比较低的水平（6% 左右）。因为如果不选择带有掩码位置的 bit 为作为输入，则可化简为精确匹配的场景，因此也不会遇到无法匹配的问题。

- TCAM 表。与 LPM 相似，根据哈希函数抽取 bit 位置是否带有掩码 (*) 位，TCAM 表项也可能会占用比原始表项空间更多的流表项数目。
- RAM (直接查找) 表。查表时以带搜索 KEY 的值为地址，直接读出 RAM 地址位对应的存储数据作为搜索结果。若搜索的数据位宽为 KEY 位，则 RAM 表的深度应为 2^{KEY} 。由于 RAM 表的深度只与 KEY 的位宽相关，RAM 表的存储依赖于连续地址空间，因此 RAM 表拆为 K 个也无法使 RAM 表的容量缩小为原来的 $1/K$ 。在第二章本文介绍过，由于使用 RAM 查表的域宽度一般不大，只有 16 位（查找端口号等），占用空间很小（不超过总资源的 0.5%），可以考虑完整

复制。其次，若 RAM 真实利用量比较小，可以用 CAM 代替 RAM 表，从而更节约逻辑资源。

4) 多级流表与流量均衡分类器

在数据包处理的过程中，假设匹配 N 个域，每个域共 M 个流表项，若只使用一张流表，则总共需要消耗 M^N 深度的流表项。一般实际应用时，会为每个域单独设立一个查找表，多个域使用多级查找表以串行流水线方式完成匹配过程。这样只需使用 $M \times N$ 深度的表项，可大大节约资源消耗量。

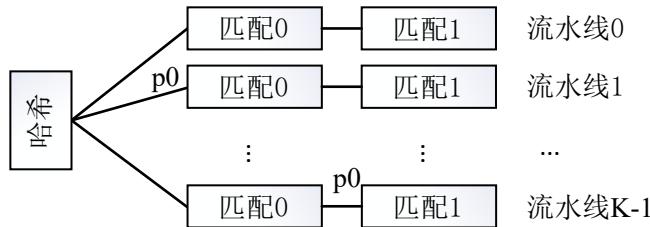


图 3-8 流表分割引发的数据包处理信息丢失问题

然而，通过第一条子流水线的数据包在匹配下一个域时，下一张流表的流表项不一定与此数据包处于同一条子流水线。如图3-8所示，数据包 $p0$ 根据流量均衡哈希函数被分配到第 1 条子流水线，经流表 0 匹配结束后，此数据包还需流表 1 继续匹配，但此时发现，处理 $p0$ 数据包对应的流表 1 不一定依然存储在流水线 1 中，也许被拆分到了流水线 $K-1$ 内。则数据包 $p0$ 无法得到后续服务。

为解决此问题，本文提出一种新的“并行查找块结构”，对流量均衡哈希函数以及多级流表之间数据传递进行的重新设计。如图3-9所示，每一个查找块代替传统查找流水线内的一个流表（匹配 0 或匹配 1）。查找块使用串行方式组成流水线。每一个查找块内都包含了流量均衡分类器。数据包首先进入分类器，分类器提取出包头域中特征位，将此数据作为哈希函数的输入值进行哈希处理得到一个 8 位 ID 数字。本文将此 ID 号码代表前文提到的流分类组份 j 的值。接下来的 RAM 表中保存了对每一个 ID 编号应该由哪一个子表查找的信息。数据包得到子表编号之后，通过一个交换网络转发到对应的子表流水线中，并完成相应域的查找-执行动作。分类器中子表的映射关系直接关系到后续子流水线的查找任务负载程度，以及子表中存储的流表项数目大小。

本结构解决了本小结前述的挑战，增加了并行处理性能，同时能够减少复制流表所耗费的资源空间。但也引入了一系列其他结构，如分类器，交叉开关以及多路执行机构。接下来本文分析这些子模块的设计复杂度以及逻辑资源开销。

- 分类器。分类器由特征提取器，哈希函数以及一个查找子表组成。特征提取器和哈希函数结构简单，特征提取器在电路中使用一个固定位选路器即可实现。哈希函数使用一些异或门电路以及少量加法逻辑可实现，资源占用可以忽略不计。查找子表是本模块消耗资源最大的，但对于一般流量，只需要将其分为 256 或 1024 (j) 类即可。其最多占用存储容量 1KB，现代通用型 FPGA 的存储资源大上百兆，因此也可以忽略不计。

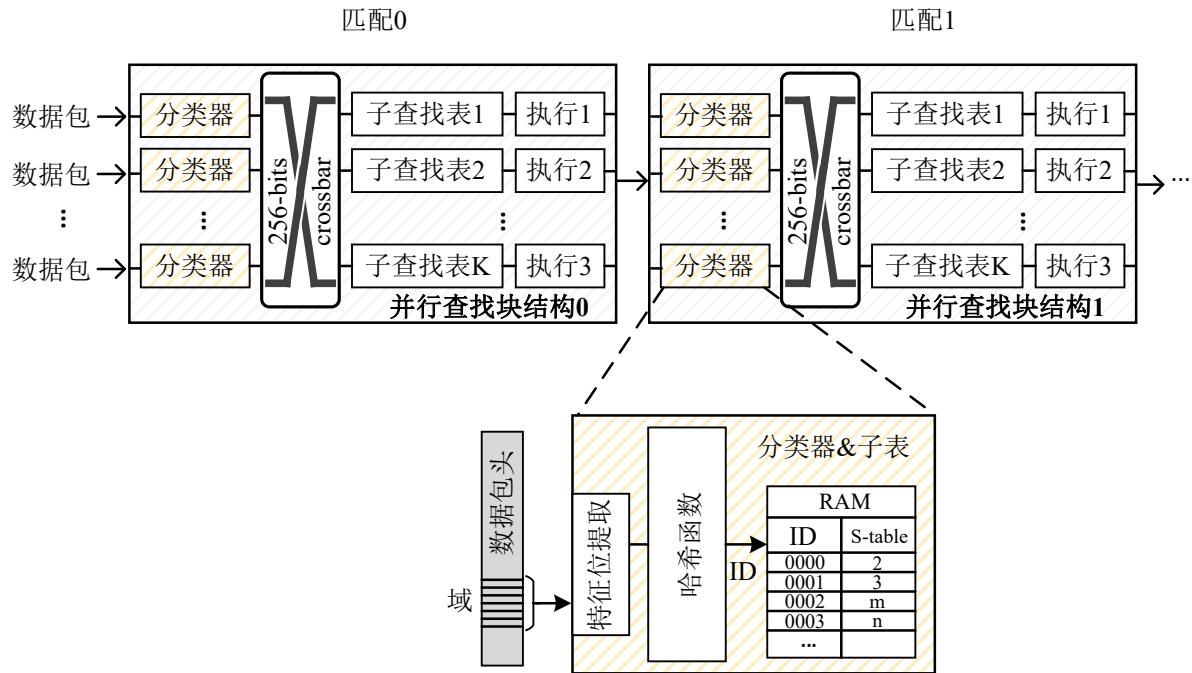


图 3-9 PL 端带有流量均衡的并行查找块结构

- 执行器。执行器内无须占用大面积存储资源，一些处理和控制功能属简单逻辑，因此执行器所消耗资源也微乎其微。
- 交叉开关。交叉开关可为 K 个输入通道与 K 个输出通道同时互相连接。是一种高性能的信息交换机制。交叉开关为每个输入与输出接口都放置了一个连接开关机构，因此一个 K 口输入输出交叉开关需要 K^2 个逻辑资源复杂度。本文在设计并行查找块时，其并行数量 K 在 4–8 个之间，因此交叉开关的资源耗费量也在可控范围内。

本文将在本章的3.7小节中给出以上几类模块的资源消耗，从数据可以看出，本文引入的模块与流表容量相比是可以忽略不计，因此设计是有效的。本文对一般结构的数据包处理流水线做了两方面的优化。在横向，每条子流水线包含了多级查找表设计；纵向，每条流水线之间的同一年级流表中流表项以分散的形式存储在子流表内。本文并不改变传统的查找表与执行机构的设计，只改变了其组织和使用方式，因此查找表依然可以使用目前工具链由高层次语言编译。因此在 PL 的编译中，本文的前端编译器只需要将流表定义转译为拆分后的结构。其他的辅助模块结构是固定的无需再修改，这也为本文的前端编译器设计工作减轻了负担。通过以上两种优化方法，本文可同时享受到并行包处理流水线带来的高吞吐量，也无需大量复制流表项，且能保证流表资源不成倍增长。

3.5 流表、流量分配问题

3.5.1 流量均衡的问题分析

对本文提出的自适应交换系统的编程涉及到两个部分，一部分是分类器子表的配置，另一部分是流表项的配置。由前文分析可知，分类器子表的配置影响到后续子流水线中流量分配，以及子表中流表项数目的占用。而且如果更改子表配置，则后续对应的子流水线流表项也许同步进行删除和添加。分类器子表的配置是根据不同流组份流量大小来确定的。根据真实流量的分析^[139]，平均情况下一个流组份的流量大小基本保持不变。因此在得到最佳配置之前，可以先通过测量的方式来对分类配置进行实时调整。接下来本文分析分类表以及流表配置过程。

在哈希函数将数据包分类到 j 个组份之后，每个组份的平均流量大小是随机的，且不均匀。分配过程的主要任务是将各个组份分配到 K 条子流水线之后使子流水线中的流量尽量保持均衡。此外，每个组份所包含的流个数也不尽相同，但每一条子流表的深度是相同的（假设为原来匹配流表深度的 $1/K$ ），分配过程还应注意不同类别的流表拆分会导致流表多份复制，且应该使得每个子流表中所分配到的流表项数目均衡，并最大限度地减少流表资源的额外消耗。

未解决上述的多约束问题，本文首先将问题数学抽象：对于图3-9中所示的一个并行查找结构。假设 N_0 是这一级匹配拆分前所对应的原始流表项数目； C_0 是原始流表容量深度； R_d 是流表的资源消耗率； N 是并行流表中流表中已消耗的资源总量； K 是流水线并行度； $C_k(k \in [1, K])$ 是每个子表的设计容量；分组 ID 号码由 $\text{HASH}()$ 函数求得，且 $j = \text{HASH}(P \text{ bits})$ ，输入数据是一个 P 位宽度的数值， j 表示了 ID 值的空间范围。

因此可以得到对于流表资源的总约束条件：

$$N \leq \sum_{k=1}^K C_k \leq C_0 \times R_d \quad (3-2)$$

假设 $D_flow[i]$ 是网络中数据包流 i 的流量大小，则有：

$$\sum_{i=1}^N D_flow[i] = 1, i \in [i, N] \quad (3-3)$$

$D_id[j]$ 是属于同一个 ID 分组内的总流量大小：

$$D_id[j] = \sum_{flow[i] \in j} D_flow[i] \quad (3-4)$$

T 为同级内各个流表之间流表项利用均衡度差异（单位：条）。由于对 FPGA 内总

资源的把控需求，可为流表多倾斜分配的资源有一个上限，可用 T 来表达。令 S 是组份 j 所组成的集合， $S = 1, 2, \dots, 2^P$ 。 Q_k 定义为每个哈希后 ID 组份的最终分配情况， $Q_k \in S$ ， $\cup Q_k = S$ ， $|Q_k|$ 表示一个子流表中组份的总个数。令 $D[k]$ 为第 k 个子流水线所承载的总数据包吞吐，则有：

$$D[k] = \sum_{j \in Q_k} D_id[j] \quad (3-5)$$

如前文（流表拆分）中分析，一个带有掩码的流表项可被拆分到不同的组份 j 中，但其中不同的组份还有可能被重新分配到同一个子流表内。因此可以将已经拆分开的掩码表项重新合成一条。定义总共被合并掉的表项个数为 V_k 。本文称待解决优化问题为“基于流量均衡的流表组合问题”（The Load-Balance-Based Table Construction Problem, LBBTC）。

定义 3.1 (LBBTC): K 为系统并行流水线数目，当 $k \in [1, K]$ ，若服从：

1. 基本约束， $Q_k \subseteq S, k \in [1, K], \cup Q_k = S; \sum_{i=1}^k C_k[i] \leq C_0 \cdot R_d$;
2. 流表空间占用限制， $\left\| \frac{Q_i}{C_{ki}} - \frac{Q_j}{C_{kj}} \right\| \cdot 100\% \leq T (i, j, ki, kj \in [1, k])$;
3. 布尔函数， $BOOL(i, j) = \begin{cases} 1 & j \in Q_i \\ 0 & j \notin Q_i \end{cases}$;
4. $G[j] = \sum_{i=1}^k BOOL(i, j) = 1 (j \in S)$ 。
5. $\sum_{j \in S} G[j] = 2^P (j \in S)$;
6. $D[k] = \sum_{j \in Q_k} D_id[i] (k = 1, \dots, K)$;
7. 优化目标 1， $F_1[k] = MAX(D[k]) - MIN(D[k]) \leq q_1$;
8. 优化目标 2， $F_2[k] = C_0 - \sum_{k=1}^K V_k \leq q_2$;
9. 帕累托最小优化， $F[k] = (F_1[k], F_2[k]) (k \in [1, K])$

对于帕累托最小优化的求解定义为 LBBTC 问题。 ◇

3.5.2 NP 难问题证明

本文证明上述问题为 NP 难解问题，而后给出一种基于模拟退火的启发式算法。证明思路中，首先引入一个经典的 NP 完全问题“平均分割问题”，随后本文通过给出此问题的退化过程，证明“基于流量均衡的流表组合问题”是一个 NP 难问题。

待解决优化问题为“基于流量均衡的流表组合问题”（The Load-Balance-Based Table Construction Problem, LBBTC），已知其数学形式化后如定义 3.1 所给出。

定理 3.1: K 为系统并行流水线数目，当 $k \in [1, K]$ ，对于待求问题帕累托最小优化：

$$F[k] = (F_1[k], F_2[k]) \quad (k \in [1, K]) \quad (3-6)$$

是 NP 难问题。 \square

证明：有用语如下：如果表达式 L_1 可以在多项式时间复杂度内归约到表达式 L_2 ，记做 $L_1 \leq_p L_2$ 。若有一多项式时间计算函数 $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ ，则可描述为：对于任意 $x \in \{0, 1\}^*$, $x \in L_1 \Leftrightarrow f(x) \in L_2$ 。

平均分配问题 (AVG_DIV): 对于一个有限集合 $S = \{1, 2, \dots, n\}$ ，重量转换函数 $w : S \rightarrow \mathbb{Z}$ ，有如下待解问题：是否存在一个子集 $S' \subseteq S$ 满足：

$$\sum_{x \in S'} w(x) = \frac{1}{2} \sum_{x \in S} w(x) \quad (3-7)$$

平均分割问题可描述如下，AVG_DIV:=

$$\{\langle S, w \rangle : S \subset \mathbb{N},$$

函数 w 是 $\mathbb{N} \rightarrow \mathbb{Z}$ 的映射关系，

存在子集 $S' \subseteq S$ 使得：

$$\sum_{x \in S'} w(x) = \frac{1}{2} \sum_{x \in S} w(x)\}$$

此经典问题 AVG_DIV 已经被证明是 NP 难问题^[140]。

接下来通过给出此问题的退化过程，证明“LBBTC 问题”是一个 NP 难问题。令 S 是组份 j 所组成的集合， $S = 1, 2, \dots, 2^P$ 。 R_d 是流表的资源消耗率； K 是流水线并行度； $C_k (k \in [1, K])$ 是每个子表的设计容量； T 个流表之间流表项利用均衡度差异； $D_id[j] (j \in S)$ 是属于同一个 ID 分组内的总流量大小：定义总共被合并掉的表项个数为 V_k ；以及两个实数变量 q_1 与 q_2 。待求问题为是否存在一个分配组合 $\{Q_i\}_{i=1}^K$ 可满足下列约束条件：

$$Q_k \subseteq S, k \in [1, k], \cup Q_k = S; \sum_{i=1}^k C_k[i] \leq C_0 \cdot R_d \quad (3-8)$$

$$\left| \left| \frac{Q_i}{C_{ki}} \right| - \left| \frac{Q_j}{C_{kj}} \right| \right| \cdot 100\% \leq T \quad (i, j, ki, kj \in [1, k]) \quad (3-9)$$

$$BOOL(i, j) = \begin{cases} 1 & j \in Q_i \\ 0 & j \notin Q_i \end{cases} \quad (3-10)$$

$$G[j] = \sum_{i=1}^k BOOL(i, j) = 1 \quad (j \in S) \quad (3-11)$$

$$\sum_{j \in S} G[i] = 2^P \quad (j \in S) \quad (3-12)$$

$$D[k] = \sum_{j \in Q_k} D_id[i] \quad (k = 1, \dots, K) \quad (3-13)$$

$$F_1[k] = \text{MAX}(D[k]) - M\ln(D[k]) \leq q_1 \quad (3-14)$$

$$F_2[k] = C_0 - \sum_{k=1}^K V_k \leq q_2 \quad (3-15)$$

帕累托最小优化函数：

$$F[k] = (F_1[k], F_2[k]) \quad (k \in [1, K]) \quad (3-16)$$

平均分割问题 (LBBTC) 可描述如下，AVG_DIV:=

$\langle S, K, Rd, C_k, T, D_id, V_k, q_1, q_2 \rangle$:

ID 组份组成了集合 $S = \{1, 2, \dots, 2^p\}$,

并行流表个数 $K \in \mathbb{N}$,

系统内流表资源冗余度 $Rd \in \mathbb{R}$,

$T \in \mathbb{N}$ 个流表之间流表项利用均衡度差异 (单位：条),

D_id 为 $S \rightarrow \mathbb{R}$ 的映射, 表示以不同 =ID 分组的流量分布,

C_k 为每个流表的容量,

在流表拆分后, V_k 表示每个流表内, 最终可被重新合并的流表项个数,

$q_1, q_2 \in \mathbb{R}$, 存在一个分配规则 $\{Q_i\}_{i=1}^K$ 满足 LBBTC 问题所描述的约束条件。}

使用描述语言定义 LBBTC 问题, AVG_DIV:=

为说明 $\text{AVG_DIV} \leq_p \text{LBBTC}$, 即 AVG_DIV 可被归约到 LBBTC 问题。令 $\langle S_1, w \rangle$ 为是 AVG_DIV 的一个实例, 可以继续构造一个 LBBTC 的实例如 $\langle S, K, Rd, C_k, T, D_id, V_k, q_1, q_2 \rangle$ 如下:

令: $S = S_1$, $D_id[i] = w(i)(i \in S_1)$, $K = 2$, $Rd = 1$, $T = \infty$, $q_1 = 0$, $V_k = 0$, $C_k = \frac{C_0}{K}$, $q_2 = C_0$, 由于 T, q_2 , 为常数, 第二个和第八个约束条件释放, 利用第 4 和第 5 个约束条件, 可得到: $Q_1 \cap Q_2 = \emptyset$, 因此约束条件变为:

$$Q_1 \cup Q_2 = S_1 \quad (3-17)$$

$$Q_1 \cap Q_2 = \emptyset \quad (3-18)$$

$$D[j] := \sum_{x \in Q_j} w(x) \quad (j = 1, 2) \quad (3-19)$$

$$F_1 : \text{Max}(D[j]) - \text{Min}(D[j]) = 0$$

$$\Leftrightarrow D[1] = D[2] = \frac{1}{2} \sum_{x \in S_1} w(x) \quad (3-20)$$

接下来说明当且仅当 $\langle S_1, 2, 1, \frac{C_0}{K}, \infty, w, 0, 0, C_0 \rangle \in \text{LBBCD}$ 时, 有 $\langle S_1, w \rangle \in \text{AVG_DIV}$:

如果存在一组分配机制 Q_1, Q_2 使得上述约束条件得到满足，则令 $S'_1 = Q_1$ ，根据约束 A-13, A-14， $\sum_{x \in S'_1} w(x) = \frac{1}{2} \sum_{x \in S_1} w(x)$ 。另一方面，如果存在一个子集 $S'_1 (S'_1 \subseteq S_1)$ ，满足 $\sum_{x \in S'_1} w(x) = \frac{1}{2} \sum_{x \in S_1} w(x)$ ，则当 $Q_1 = S'_1$ 且 $Q_2 = S_1 - S'_1$ ，显然约束条件 A-12 至 A-15 全部满足。所以 AVG_DIV 归约到 LBBTC，LBBTC 同样为 NP 难问题。■

3.6 资源优化问题的启发式算法设计

本小结将给出一种启发式算法，使前文提到的 NP 难问题在可接受时间内得到一组有效解。简单解思路为暴力搜索所有 Q_k 的可能组合，来寻找一组均衡的流量分布结果。对于 N 个 ID 组份的计算搜索次数可以被估计为：

$$T(N) = K^N \quad (3-21)$$

其中 K 为并行子表的个数，当 $N = 16$ 时，计算搜索数约为 10^{10} 数量级，以目前服务器 CPU 计算能力约须消耗 10 秒钟才可找到精确解。然而在真实场景下，一般须设置 $N \geq 64$ （为使各个流水线中流量更均衡），搜索空间会陡增至 10^{38} 数量级，甚至可能将 N 设置为表达 256 个 ID 组份之多。

本文提出一种基于模拟退火方法的启发式算法。在 PL 的多级流表中，每一级流表组合机制的结果空间为初始集合 S 的子集 Q_k 。模拟退火算法的状态转移探索方式为下列情形二之一：(1) 从集合 Q_k 中随机地移动一个元素到任意其他的集合 $Q_{k'}$ ；(2) 交换任意一组(2个)集合 Q_k 中的两个元素。首先，在每一个启发式搜索状态内，须保证 Q_k 内元素的有序性。如果流量都很均衡则哈希后的 ID 组份流量也会均衡，但考虑到真实网络流量的“80/20”分布定律，哈希后流量分布也呈现巨大差异性。差异性体现在两方面：第一是流量的不均衡性，第二是流数目分布的不均衡性。因此本文将解空间 (Q_k) 内的元素分为两个属性，一部分是流量巨大的 ID 组份，另一部分是小流所代表的 ID 组份。在第二种转移探索方式中，各集合 Q_k 内同属性元素做互相交换。例如，做集合 Q_1 和 Q_2 元素交换时，只选择每个集合里符合大流属性的 ID 组份间进行交换。大流组份不会与小流组份进行交换。因此退火算法可以避免大量的无效探索次数。

本文定义启发式算法的能量方程（评价函数）如下：

$$E_1(n) = \frac{\text{Max}|D[k] - \text{avg}(D[k])|}{\text{avg}(D[k]) \times 100\%} \quad (3-22)$$

$$E_2(n) = \frac{\sum K(C_k - C_0)}{C_0 \times 100\%} \quad (3-23)$$

模拟退火的核心优化思想是，接受一个新探索状态的条件是满足新的能量低于当前能量。在本文的启发式算法中，进入新阶段的移动条件是：

Input: 流量统计结果 $D_id[i]$

Output: 初始化解空间 Q_k

```

1  $Q_k = \phi; D[k] = 0; i, j, k = 0; i, j, k \in \mathbb{Z}; // \text{初始化}$ 
2 对  $D\_id[i]$  按降序排列, 记为  $\{Sid[1], Sid[2], \dots, Sid[2^P]\}$ 
3 for  $i \text{ in } [1, 2^P]$  do
4   for  $j \text{ in } [Sid[1], \dots, Sid[2^P]]$  do
5     对  $D[k]$  按升序排序, 获取各小流量 ID 组份, 记为  $\{Sd[1], \dots, Sd[k]\}$ 
6     for  $k \text{ in } [1, K]$  do
7       if  $|Q_{Sd[k]}| = Min |Q_i|, i \in [1, K]$  then
8          $D[Sd[k]] += D[Sid[i]]$ 
9          $Q_{Sd[k]} \cup = \{Sid[i]\}$ 
10        break
11 // 获得了流表容量占用均衡的一个排列:  $\{Q_k, D[k], k \in [1, k]\}$ 
12 重新回到按照  $D\_id[i]$  原始的降序排列, 以下按照流量公平优先的方式分配:
13 for  $i \text{ in } \{Sid[1], \dots, Sid[2^P]\}$  do
14   find  $D[k_i] = Min(D[k]); k, k_i \in [1, K]$ 
15    $Q_{k_i} \cup = \{Sid[i]\}$ 
16    $D[k_i] += D[Sid[i]]$ 
17 // 获得了流量均衡的一个排列:  $\{Q'_k, D'[k], k \in [1, K]\}$ 

```

算法 3-1 LBBTC 算法初始化

$$\begin{cases} \Delta E_1 = E_1(n+1) - E_1(n) < 0, \\ \text{and,} \\ \Delta E_2 = E_2(n+1) - E_2(n) < 0 \end{cases} \quad (3-24)$$

另外, 当满足移动条件时, 算法最终以概率 $P = e^{-|\frac{\Delta E_1 + \Delta E_2}{t}|}$ 接受移动推荐。温度参数 t 初始值为 100, 并且在之后迭代中以指数函数形式 ($t(n+1) = t(n) \times 0.99$) 下降到 1。如算法3-2所示, 最终停止搜索的跳出条件为:

$$\begin{cases} E_1 < q_1 \& \& E_2 < q_2, \\ \text{or,} \\ t = 1 \end{cases} \quad (3-25)$$

如算法3-1描述了退火算法初始化的过程, 算法3-2描述了模拟退火算法根据初始化值进行元素交换、退火、最后退出的过程。

Input: 初始化后的两组解空间 Q_k 、 Q'_k 及其对应的流量统计和 $D[k]$ 、 $D'[k]$

Output: 经过元素交换收敛后的答案解空间 Q_k

```

1 define:  $\Delta(D, K) = \sum_{k=1}^K \left| D[k] - \frac{100\%}{K} \right|^2$ 
2 if  $\text{Max}(|Q'_k|) \leq C_0 \cdot R_d / K$  then
3   return  $\{Q'_k, D'[k], k \in [1, K]\}$ 
4 else if  $\text{Max}(|Q'_k|) > C_0 \cdot R_d / K$  then
5   set  $C_{k_1} = 2 \cdot C_{k_1}$ 
6   while  $C_{sum} > C_0 \cdot R_d$  do
7     find: another  $C_{k'}$  and
8     set:  $C_{k'} = \frac{C_{k'}}{2}, k' \in [1, K]$ 
9   while  $|Q'_k| > C_k$  do
10    pop: a ID group from  $Q'_k$  to  $Q'_{k_1}$ 
11 计算:  $\Delta(D, K)$  and  $\Delta'(D', K)$ 
12 if  $\Delta > \Delta'$  then
13   return  $\{Q'_k, D'[k], k \in [1, K]\}$ 
14 else
15   while  $Times--$  do
16     交换  $Q'_k$  和  $Q'_j$  内的 ID 组份
17     找到最小的  $\Delta'$  并且重置  $Q'_k, D'[k]$ 
18   if  $\Delta' \leq \Delta$  then
19     return  $\{Q'_k, D'[k], k \in [1, K]\}$ 
20   else
21     return  $\{Q_k, D[k], k \in [1, K]\}$ 

```

算法 3-2 LBBTC 启发式算法

下面对本文提出的并行查找块结构进行一般化描述，如图。并行结构查找块是 PL 中承载用户定义需求的核心逻辑框架，一种可扩展形态如图3-10所示。总体看系统中一共有 $m(m \geq 1)$ 个处理流水线，每个流水线为一条完整的并行查找块结构。在每个流水线中总共有 $n(m \geq 1)$ 级块结构。图中红色正方形框圈出的是流水线中的并行查找结构（基本处理单元，Basic Processing Unit，BPU）。在每一个 BPU 中，都有并行的“处理引擎”，本文中将其抽象为“数据-执行”（Data-Action）模型，其可以为查表结构，也可以为自定义计算结构。每一个 BPU 的输入（数据流）个数与执行引整个数相同，但 BPU 对外的输入输出个数不一定相同。假设第 i 级 BPU 的输入有 K_i 个，输出有 K_{i+1} 个。在第 i 级中，流调度模块（流量均衡查找模块与交叉开关）将 K_i 个输入数据流尽可能

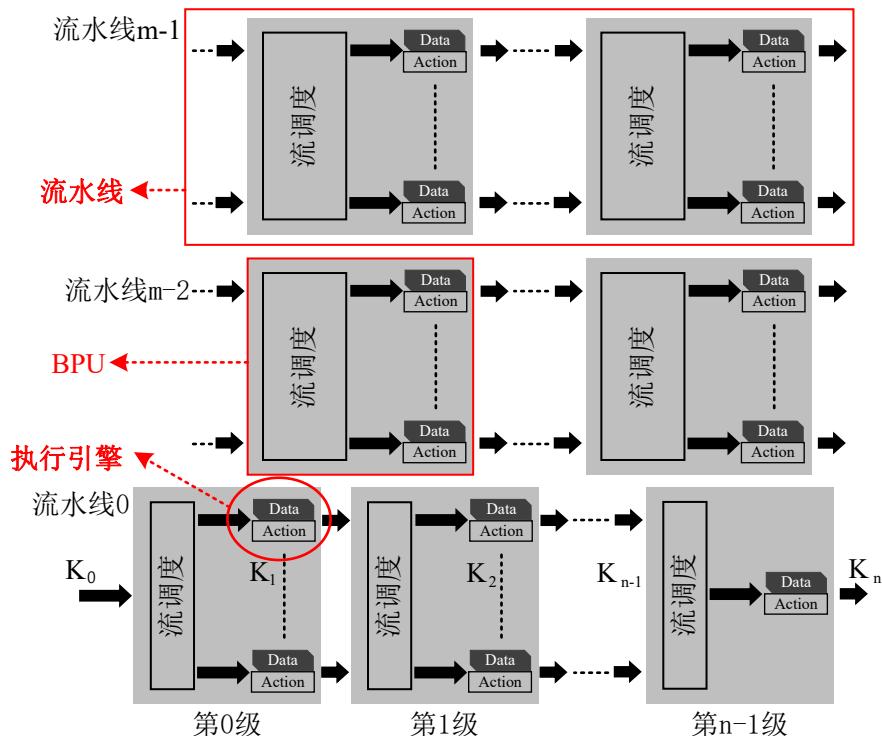


图 3-10 PL 端并行处理流水线一般模型

均衡地分发到 K_{i+1} 个执行引擎中。执行模块的设计以及系统参数 (m, n, K_i 等) 可由高层次综合工具, SDNet 或 HDL 语言来定制。每个流水线之间是完全并行的, 并无数据交叉, 每个流水线内部存在数据流量交叉。因此在计算流表与流量分配问题时, 可利用算法3-1、算法3-2针对每一条流水线独立计算。

3.7 系统开发以及测试

对前文提出的系统进行评估, 本文完成了 4000+ 行硬件描述语言 HDL 代码, 在基于 Xilinx ZC706 开发板上对 “自适应交换” 系统做了原型实现。完成内容包括了 SS 功能, 以及 PL 端的三个用户定义测试例子。在 SS 中, 本文将一个发包模块集成在 “自适应交换” 系统中, 以方便大规模地进行流量测试。本文选用了 5 种流量激励: 其中两条真实流量 (1) 由 ISP 骨干网捕获 (WIND 流量), (2) 基于基站通信的捕获流量 (LTE 流量), 三条模拟流量 (3) 服从指数分布流量, (4) Pareto 分布流量以及 (5) 平均分布流量:

1. WIND 真实流量。此流量由位于新西兰的接口速率 3.7Gbps 的 ISP 路由端口捕获而成。流量包括了 4 十亿个数据包, 持续时间 60 分钟。
2. LTE 真实流量。此流量捕获于校园内的移动信号运营商 4G 基站, 捕获数据时保护用户隐私, 隐去了数据包数据信息, 只保留了包头信息和包长度信息。在回放数据时, 本文使用随机数字填充数据包内容。此流量共计 1 亿个数据包。
3. 指数分布流量。流量中每条流的大小服从指数分布, 指数分布是一种长尾分布, 小流数目众多。本文将指数分布中的最大流量设置为 100Mbps, 尺度参数 $\lambda = 30$ 。

指数分布流量平均流大小为 3.33MB。

4. Pareto 分布流量。流量中每条流的大小服从 Pareto 分布, Pareto 分布是一种厚尾分布, 形状参数 $\alpha = 30$, 位置参数 $\beta = 1.05$ 。本文设置最大单流流量为 100Mbps, 平均流大小为 8.6MB。
5. 平均分布流量。流量大小服从 1KB 到 100MB 的平均分布。平均流量大小为 50MB。

本文通过 300+ 行 python 代码实现了 PL 映射优化算法, 其运行在通用服务器中 (Dell R620 主机, 8G 内存, 2.80GHz 主频, Ubuntu16.04)。本文利用 PyPy 工具链加速了 python 程序性能优化。

为提现“自适应交换”系统的灵活性, 本文实现了三个网络功能应用场景。

3.7.1 拥塞控制

NDP^[135] 可保障在交换机内拥塞丢包时“小批量”数据包的转发延迟。当数据平面拥塞时, 通路缓存空间占用率会上升。对于短会话数据包, 等待每一个阶段的缓存排队将导致会话完成时间成倍增长。此外, 当数据平面内发生丢包现象, 按目前的 TCP 协议将使用很长时间才能被动发现, 导致重传时间过久, 通信时延成倍增长。NDP 则是一种数据平面内的拥塞控制机制, 可在数据平面内实时发现丢包事件, 并快速双向通报(向源与目的同时通报), 实现快速重传, 与快速窗口收敛。NDP 是一类“事件驱动”处理算法, 在通用可编程数据平面内只支持包触发行为, 因此无法直接应用。为在“自适应交换”系统中实现 NDP, 本文利用 MMU 在 SS 端实现逻辑输出队列, 在 PL 端实现了两个 NDP 执行集: (1) 检测队列深度是否到达拥塞域值; (2) 发送精确信息控制来调整包大小。前端编译器将此执行集编译到 BPU 和流水线内, 同时实现功能落地和性能扩展。

3.7.2 网络测量

前文介绍 DISCO^[105] 是一种高效率高精度的网络流量测量算法。前文已经在基于网卡端的可编程硬件中详细阐述了 DISCO 的硬件优化方法。由于目前商用的可编程数据平面 (P4/非 P4 交换平台) 同样缺乏计算指令集以及灵活的判断分支方法, DISCO 依然难以使用目前技术在交换核心节点实现。在“自适应交换”系统中, 本文将实现交换机内 DISCO 硬核。传送进入 PL 端的元组数据 (metadata) 须包含, 包长度, 流标识号 (flow ID)。PL 端将流统计结果暂存于板载高速内存。

3.7.3 有状态防火墙

本文利用 PL 的可扩展性在“自适应交换”系统中实现了一套有状态防火墙转发引擎, 目前有状态转发在商用可编程交换机内依然无法^[141-142] 实现。状态引擎为待过滤流保存连接状态。须实现一个组合表, 由两个子表构成: (1) 基本的查找表实现“匹配-执行”操作; (2) 存储一系列状态转移图的信息。当数据包到达组合表后, 执行器依据当前状态标志以及匹配结果对数据包进行下一步处理。之后根据下一状态查找表

更新基本表以及目前状态标志位。

3.7.4 资源消耗

表 3-2 各组件性能以及处理时延

组件	最大运行频率 / MHz	阻塞时间 / 周期	时延 / 周期	吞吐 / Gbps
MMU(1536×4096 字节)	235	3	2	147
L3 包头解析	400	1	6	125
分类器 & 256 查找子表	400	1	5	125
4:4 交叉开关	303	1	2	378
匹配表	CAM 4k 64bit	176	1	3
	LPM 4k 32bit	191	1	20
	TCAM 1k 32bit	171	1	22
实用场景	NDP	312	1	8
	测量	265	1	3
	防火墙	416	3	7

表 3-3 各组件 FPGA 内资源消耗情况以及所占比例

组件	LUT	LUT RAM	FF	BRAM	DSP
MMU(1536×4096 字节)	626/0.29%	272/0.39%	212/0.05%	7.5/1.38%	0
L3 包头解析	242/0.11%	16/0.02%	862/0.20%	0	0
分类器 & 256 查找子表	40/0.02%	16/0.02%	36/0.01%	0.5/0.09%	0
4:4 交叉开关	252/0.12%	88/0.12%	298/0.07%	1/0.18%	0
匹配表	CAM 4k 64bit	760/0.35%	1937/2.80%	1693/0.39%	15/2.8%
	LPM 4k 32bit	861/0.39%	813/1.2%	2466/0.57%	23/4.2%
	TCAM 1k 32bit	36434/17%	23022/33%	35068/8.1%	3/0.55%
实用场景	NDP	271/0.12%	0	8/0.01%	7/1.28%
	测量	933/0.43%	20/0.03%	599/0.14%	8/1.47% 1/0.11%
	防火墙	37/0.02%	0	232/0.05%	8/1.47%

表3-2和表3-3展示了“自适应交换”系统关键部件的 FPGA 实现特性以及资源消耗。本文记录了实现模块的最大运行频率，以及实现流水线中的阻塞情况和模块内部流水线时延，最后得出了理论状态下的最高吞吐性能。由于 SS 端须依赖 ASIC 实现，因此部分模块性能还会显著提升（如 MMU 将会有十到二十倍增长空间）。

3.7.5 控制平面计算量

PL 端系统高速运行依赖于流量的负载均衡，但实际流量依然存在平均速率值的上下偏离。因此实时系统须对流表分配进行重排。重排可重新使流量均衡，但过程中也会导致大量的控制信令占用控制通道带宽。因此系统在两种情况下会对流表分配进行重排：(1) 用户增加或修改流表项；(2) 等待一定时间 $T(t)$ 之后。 $T(t)$ 可根据流量分布定义，也需要考虑安全通道的带宽容量。

1) 流表分配运行时间。

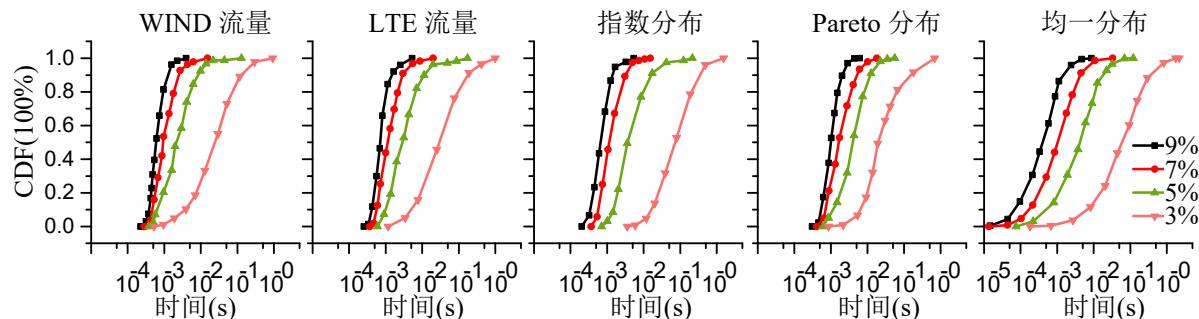


图 3-11 流量均衡算法计算性能 (时间)。在不同的均衡性 ($\delta=9\%, 7\%, 5\%, 3\%$) 需求下为 $K = 8$ 条子流水线在不同流量下的计算时间分布

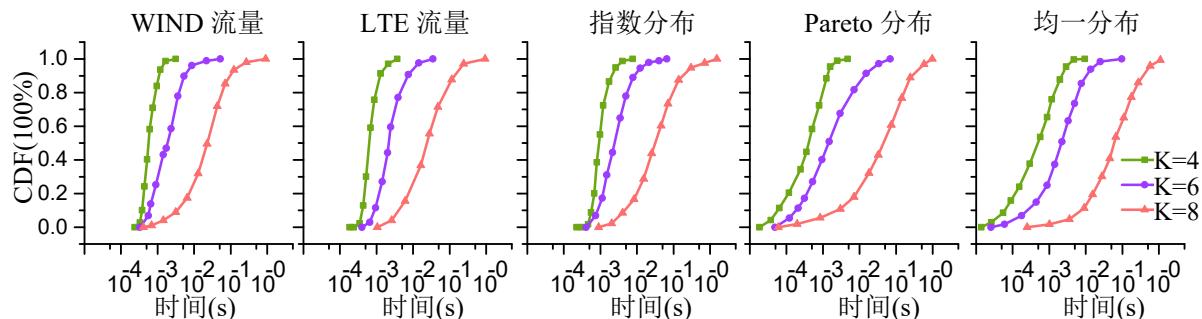


图 3-12 在不同子流表数目下 ($K = 4, 6, 8$)，且 $\delta=3\%$ 时的计算时间分布

本文在普通服务器中运行控制平面启发式算法，为了验证算法实时性，本文统计未配置状态下的数据平面（无转发逻辑，无实时统计结果）得到配置的时间。系统以一种随机状态开始接收流量，并计算每个 ID 组份的流量大小，在一定时间段之后，控制器重新计算新的流量均衡的配列方式并重新下载入数据平面。实验使用前文提到的流量数据。本文以精度 δ 来衡量流量均衡度， δ 定义为每个子流水线的流量值的算数平方差，因此 δ 值越小，均衡性越好，也越能发挥系统的最大性能。

在系统启动须先定义 δ 的可接受域值，本文设定 $K = 8$, $J = 256$ ，如图3-11所示，当设定均衡度为 5% 时，可保证在 75% 的情况下计算时间小于 0.01 秒。作为对照，如图3-12本文测试了当子流水线数目 (K) 不同时计算时间的分布。当并行度增大时，显然算法的收敛时间会增加，在本实验的三个测试功能中，超过 90% 的计算时间都小于 0.5 秒，并且能够以 100% 置信度保证计算时间小于 1.12 秒。

2) 控制通道带宽消耗。

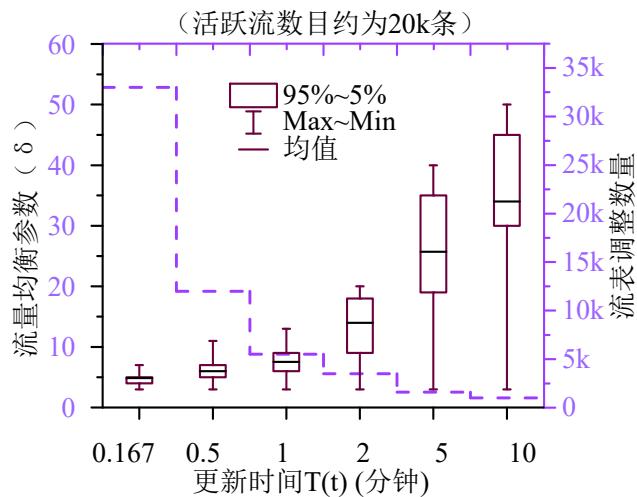


图 3-13 控制信道消耗量与流量均衡性之间的折中，实验须使用真实流量，已获得数据包之间的时间

控制器判断流量均衡，并决定是否对其进行重新调整。本文通过对真实流量的测量，对在不同均衡需求以及不同控制通道带宽使用量进行测量。在实验中每使用一个控制数据包，也定义为增加一次流表修改次数。如图3-13所示，在真实流量（WIND 流量）下，可以看出选择更新时间 $T(t)=45$ 秒或 90 秒，平均可以保证流量均衡程度在 9% 以内。若每个控制信令包长度 128Bytes，则总共消耗信道带宽不超过 3KBps。

3) 吞吐性能提升。

如图3-14所示，画出了吞吐随处理引擎并行度的变化趋势。计算处理性能时，须依照运行频率，数据总线宽度，以及真实流量数据包的包长度（平均约 650 字节）。可看出处理吞吐基本与并行数量呈线性正比。三种应用在 PL 中的总共处理时延为 $0.136\mu s$, $0.142\mu s$, $0.130\mu s$ (测量, 防火墙, 拥塞控制)。

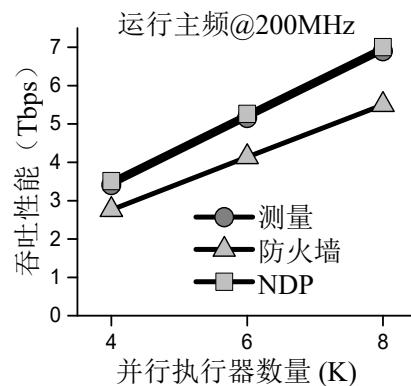


图 3-14 BPU 中不同并行度对总体性能的提升效果，FPGA 主频 200MHz

3.8 本章小结

本章首先分析了目前各类网络可编程数据平面的缺陷，为增强数据平面转发设备的网络随路计算能力，提出了一种自适应交换系统（Adaptive Switch Architecture, AS），包括编程模型（3.3小节）以及硬件原理（3.4小节）。AS 的核心思想是保留 ASIC 交换

芯片中的交换网络结构以确保高性能，但将网络数据可编程性卸载到 FPGA 可编程硬件中。接着利用前文4.3.3小节提出的设计原理，在3.5小节阐述了如何优化了 FPGA 内部的逻辑资源，证明了其配置方法属于 NP 难问题，而后给出了一种高效的启发式算法（3.6小节）。本文在 AS 架构中实现了三种难以在目前交换系统中实现的例子，集中体现了 AS 架构可为目前的交换芯片带来极高的灵活性（3.7小节）。此外，本文设计了资源消耗可控的多级并行流水线，并使以上三个应用达到了数个 Tbps 的吞吐性能。虽然 PL（FPGA）端的接口吞吐性能依然无法与 ASIC 相比，但是我们不需要将完整的数据包送入 PL，并且不是所有的数据流都需要被用户定义逻辑处理，因而可以极大缓解二者之间的性能鸿沟。综上，AS 架构的性能已经达到了业界芯片标准水平，与其相比具有很高的竞争力。

4 端侧网络在网络测量领域的可编程抽象方法

4.1 引言

为应对端侧网络对复杂可编程网络性能支持不足的问题，本章提出一种针对网络测量领域的可编程处理器。基于被动触发的网络测量系统是一种典型的测量实现技术，测量过程一般分为 1) 捕获, 2) 统计, 3) 存储三大部分，这三点是影响测量系统性能的关键因素。在网络中，中心节点网络通常对吞吐性能的需求大于对编程灵活性的需求。端侧网络不同于中心节点网络，端侧网络对性能需求较低，因而有更多的资源空间来实现更加灵活的网络系统。目前开发人员可以将所有的网络功能操作都利用服务器 CPU 来实现，并且获得非常大的可编程灵活性，但随着时间推移，数据中心服务器网络接口容量快速增长，拥塞控制、流量工程、网络监管等功能在现代化网络管理中占据重要的位置，处理网络数据的服务越来越耗费主机 CPU 的计算资源。服务器应用场景以及商业需求慢慢变化，纯粹利用 CPU 完成网络可编程功能，已经变得越来越不现实。其原因主要总结为以下几点：1) 虚拟化云计算的发展，导致服务器集群中主要的可售卖资源是以 CPU 为核心的算力设备。2) 网络流量规模快速增大，多核服务器中可以容纳更多的虚拟机资源数量，这导致服务器对网络的吞吐性能需求快速增加，目前已经到 100Gbps 以上。3) 大规模的网络通信处理，以及服务器内部虚拟机间转发，过多地占用了 CPU 计算资源，导致云计算供应商高利润的可销售部分被单一价值的网络通信占据了大半。

同时，为解决复杂网络功能软硬件协同优化时的开发复杂度，本章提出一种“统计 + 触发”的测量可编程模型，在数据平面的硬件内，对所有流进行统一的测量，然后上层软件选择感兴趣的计数值以及触发方式，共同构建起诸如拥塞控制、流量工程、网络监管的不同网络功能，同时也提高这些网络功能的性能和效率。表4-1总结了一些常见的应用场景以及其所需要的“测量”和“触发”方式的需求，例如实现 DDoS 攻击检测应用，一般需要发现不同的流，则需要对未识别的流进行匹配并记录总流数目，同时设置时间触发窗口来判断新流的增长率以供上层应用判断是否有大规模攻击现象发生。

表 4-1 不同网络应用场景下所需要的测量模式以及触发模式

应用场景	测量模式				触发模式	
	流匹配	tx/rx 流速	TCP 丢包	每包触发	时间触发	域值触发
DDoS 检测	✓	-	-	✓	✓	-
防火墙	✓	-	✓	✓	-	-
拥塞控制	✓	✓	✓	✓	✓	✓

在之后小节中分析了主机侧网络卸载软件功能的必要性以及可编程逻辑器件的适用性，并为后续设计测量的存储优化算法提供理论分析方法。本章在硬件内实现了一

种基于 DISCO^[105] 算法的非线性的压缩统计算法，并在第4.5.4小节证明本文提出的基于硬件的压缩算法是一种无偏估计过程。最后本章陆续解答本章研究内容下的两个问题：如何利用可编程网卡实现高精度、高性能保障和高能源效率的网络功能硬件卸载？提出测量功能可编程抽象、合理部署和划分任务是本文要解决的问题。

4.2 问题背景

本章就数据平面的可编程灵活性方面做继续探讨，目标是实现一套具有资源协同优化、方便使用的端到端网络系统。目前已经有一些灵活的计算任务可以放到基于多核网络处理器的智能网卡内，网络处理器本质与 CPU 相同，拥有极强的灵活性，但受到计算存储架构以及多核间通信带宽有限的影响，其性能发挥不稳定^[45-46, 106-107]。本文注意到，在数据中心内，网络的管理很复杂，需同时考虑流量控制，拥塞控制，访问控制，攻击探测等诸多网络功能的实现。目前可编程网络领域缺乏对以上偏向底层的网络功能的统一抽象，开发人员需要独立实现功能细节，重新开发某些通用模块，耗费大量的人力物力。

网络测量是实现上文功能的底层核心方法，面对高速网络的测量系统在性能方面有极大的需求^[101]。流量控制，拥塞控制需要利用网络流量实时速度以及平均包大小等相关信息，访问控制和攻击探测也需要用到流触发的反馈测量机制^[102]。目前在可编程数据平面中，支持可编程实现的网络测量技术限制于以基于 CPU 或网络处理器的平台。其性能受到 CPU 架构访存与计算能力的制约，每核心单独处理统计量能力在 10Gbps 以内^[103-105]，相比于通用服务器目前上百 Gbps 的吞吐需求，显然存在比较明显的性能弱势。一方面由于目标流量数目庞大，基于 CPU 架构的快速缓存空间不足，多级存储结构带来复杂的更新机制并且受限于外存储的非连续地址访问速率 ($<1\text{Mpps}^{[45-46, 106]}$)，性能表现力不稳定^[107]。另一方面，CPU 计算架构过于通用化，反而在处理简单重复的重任务时无法发挥优势，尤其针对流式数据处理，CPU 无法建立起可编程的高效率的流水线结构，严重制约了网络功能处理能力的可扩展性^[108]。尽管目前发展出一系列 RDMA、DPDK 等基于网卡的网络数据包快速搬运架构，但它们只针对于块数据的传递进行了优化，对于需要“每包处理”的任务依然没有良好的对策。

现场可编程门阵列（FPGA）是一种灵活性可以与软件媲美可编程硬件，性能和效率与专用硬件接近，但其硬件开发难度大。现代高速云架构依赖于每个专用硬件（ASIC）网络节点的支持，随着网络功能需求多变与复杂化，ASIC 类型的网络处理芯片并不能解决所有可编程场景需求。业界已经开始将网络堆栈向基于 FPGA 的网卡中卸载^[143-145]。为了推广可编程硬件，学术界牵头推出了基于 FPGA 的智能网卡开源项目 NetFPGA^[62]，龙头企业 Xilinx、Intel 等也纷纷推出了基于 MPSoC/FPGA 的自适应计算加速平台 Alveo^[146-147] 系列智能网卡和 N3000^[148] 系列网卡。目前，基于 FPGA 的可编程数据平面已经广泛应用在 5G 接入边缘网络^[149]、数据中心计算存储^[150]、核心网络低延迟加速器^[151] 以及高性能高可靠性高安全性的数据中心防火墙^[152] 加密通信^[153] 等领域。FPGA 一般用硬件描述语言 Verilog、VHDL 等开发。一个合格的硬件工程师的培

养周期要远大于软件工程师，这也是目前网络领域硬件卸载最难所在。为了解决这种不足业界也推出了类似 C 语言的高层次综合工具（HLS^[154]）。但使用这类工具必须学习 1000 多页的开发文档^[155]，而且并不是所有代码都可以直接被工具转译，(1) 也需要考虑硬件细节，合理修改设计以降低 FPGA 内部资源消耗；(2) 需要自主决定并行区块；(3) 需要在代码中融入这种编译器的特性标记字符。总体来看并没有从本质上改善对硬件编程的困难程度。除此之外，由于在 FPGA 中复杂逻辑对并行总线宽度的频率敏感度高，一个大型工程时序逻辑的处理主频一般不会超过 200MHz。即使每个时钟节拍都可以处理一个数据包，那么 FPGA 流水线在处理最小包时的最高吞吐量也只有 134Gbps^①，这对 FPGA 应用在核心网包交换场景形成了瓶颈。

4.3 系统架构分析

4.3.1 软件与专用硬件处理器的特性分析

CPU 通过循环取指令等操作，完成通用的可计算任务。计算的数据通常有前后依赖关系，CPU 对于此类计算效率较低：由于中间计算状态在计算完成之后必须放回数据存储区，而下次重新拿回状态数据又需要再次搬运，数据在计算核心与存储池之间多次往返对计算最终结果是无意义操作。而即使使用众核处理器也无法优化此过程，虽然众核处理器可并行计算多个任务，但同一时刻每个核心与其他核心处理内容并无关联。对于前后有依赖关系的处理，并行并不能加速其中一组数据的处理进程。在网络领域亦是如此，网络数据包到达密度很大，留给每个包的处理时间很有限，然而通常一个 CPU 无法在如此短时间内真正处理完一个包的触发计算。

1) 软件处理延迟大。

目前高速网络处理器可以设计为核间流水线模式，每个核只处理固定的一步计算。当一个数据包触发的计算包括多个前后依赖计算时，人们使用多个核串行处理这组计算。即每个核心领取一个固定的快速处理任务，这样每个核都可以以最快的速度处理完当前步骤，然后交给后续核心继续处理。这样数据包的处理吞吐其实就可以达到某一个核心的最大速度，而处理延迟则是这些核心之间传递完整一次的时间。不难发现，数据在众核之间搬运会遇到访存时间过长、访存请求冲突等现象。虽然使用 CPU 可以获取很大的灵活性，但这种方式进一步增大了每个数据包处理的时延。如今在云端加速 AI 计算的场景下，高频次小包、数据快速到达的需求越来越高，这使得用 CPU 处理网络数据包造成很大的云计算性能瓶颈。

2) 软件处理时间精度低。

同时，由于共享内存与总线架构，数据在 CPU 核心、存储池之间搬运会造成随机的请求冲突、甚至计算等待，这将中断顺序串行处理节拍。由于这条处理链中的处理速度会受到这条处理链最慢处的制约，因而处理链的性能表现总以最慢的瓶颈向外表现。软件的处理受到操作系统的指挥，操作系统一般会划分时间片区来分配给每一个待处理的任务。虚拟化的操作系统内有很多任务进程，操作系统会划分很多时间片区。

^① $134\text{Gbps} = 200\text{Mpps} * (64+20) * 8\text{bits}$

这也进一步降低了 CPU 的专用任务处理性能，还带来处理时间精度不足的现象。后续第4.4.2小节将详细展示此论点。

3) 软件功能向硬件卸载。

通过卸载优化软件功能应保证硬件可以完成相同的工作任务，但能收获更高的计算效率更高以及更快的运行速度。硬件功能卸载已经广泛使用在计算机领域。早期，开发人已经把音视频处理从 CPU 完全卸载到图形处理器和音频芯片中。此外还有一些例子：数据包校验计算卸载到网卡，传输大宗数据时使用 DMA 控制器，在硬盘控制器中做数据错误恢复。事实上，从 CPU 中卸载出来的功能都比单纯由 CPU 执行能源效率高很多。但本文并不是强调 CPU 无用论，因为 CPU 是整个服务器中最具有灵活性的器件。卸载任务应该是简单、机械的。CPU 擅长于做可编程、花样繁多的算法，应部署具有很高的难以替代性的工作。向硬件卸载软件功能也存在一些劣势：

- 增加了软硬件开发人员的工程任务量。
- 增加新的硬件，使系统结构复杂。
- 设计制造新硬件，增加研发周期、增加使用成本。
- 由于大量装配特殊器件，系统可移植性降低。

对于灵活多变的网络流量的处理任务，大多为分步骤且数据包信息前后依赖关系高。前文提到，如果使用基于 NPU 的可编程网卡，虽然灵活性可以被满足，但依然存在性能问题（单核性能低，单个数据包处理延迟大）。使用基于可编程硬件（FPGA）的智能网卡卸载网络功能，可以有效解决上述问题。表4-2展示了使用 FPGA 作为卸载功能目的实体与 CPU 和 ASIC 对比的相对优劣点。虽然 FPGA 完全有能力完成分支多且复杂的决策算法，但相比于 CPU，在 FPGA 上实现将付出更多编码时间，不如直接使用通用处理器经济收益好。另外相对软件而言，FPGA 的处理过程具有精确无抖动（jitter-free）的时序效果。虽然独特优化的软件代码可以近似达到这种效果，但它依然严重地受到调度器、中断、不同处理路径的影响而产生抖动。

表 4-2 FPGA 的优势与劣势

FPGA 所擅长的处理内容	不适宜使用 FPGA 处理的任务
流式数据通路	多决策算法
并行算法	递归
低延迟、确定延迟系统	依赖于其他大型软件库
与硬件接口紧邻	急速变化的浮点数学计算过程

4.3.2 测量系统软硬件架构

根据上述软硬件特性分析，如图4-1所示为本章提出的测量可编程处理系统架构图。为了保证网络功能的时间精度以及吞吐性能，测量处理器在硬件内集成了定制的数据报文的捕获与发送结构，处理器收到数据包后将数据包镜像流量送入统计模块并在统计表中储存统计结果。处理器中包括了一个可编程的触发机制模块，根据不同的触发

方式对不同的流统计信息进行上报，统计信息主要包括：数据包个数，数据容量，触发方式包括了：包触发，时间触发以及统计量域值触发。由触发方式以及统计信息还可以计算出更多的相关性信息，例如流量速度，TCP 未确认数据量（Bytes in flight）等。

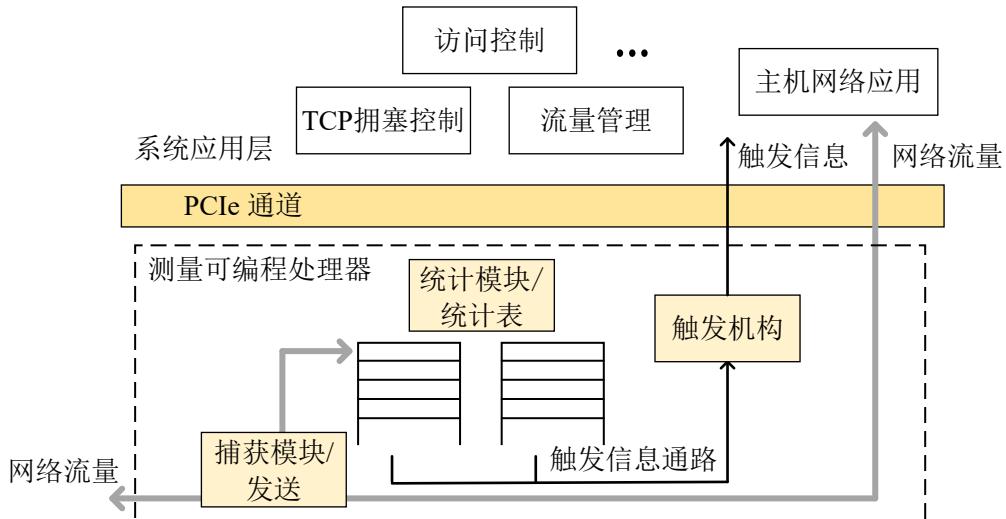


图 4-1 主机端网络测量可编程处理器的系统架构

本文利用基于 FPGA 的智能网卡实现了测量可编程处理器，网卡通过 PCIe 通道与主机连接并通信，捕获后的网络流量直接送往主机内的网络应用进程。而触发信息则上报给各类网络应用，网络应用只需要接收需要的信息即可完成各种应用的部署，相较于基于软件的应用实现方式，省略了计算繁重的统计过程，可以达到释放宝贵的 CPU 算力的目的。

4.3.3 计算流水线的优化设计方法

在高性能的大数据领域，批处理技术已经可以满足非常多的使用场景，但随着数据时效性价值提升，一种流式计算概念被提出。流式计算是实时的，常驻的，与 Hadoop, MaxCompute, Spark 等系统不同（它们是离线批处理），流式计算对延迟属性要求更为苛刻。而目前在大数据和高性能网络中的处理就更需要考虑如何利用 FPGA 来加速流式计算的需求。

1) FPGA 编程抽象。

基于 FPGA 的可编程逻辑，可以完成很多种计算。在软件思维转向硬件化的时候，如图4-2所示，给出了三种并行化加速方法：同一种功能 ($F(x)$) 的复制展开并行；一组数学计算的内部隐含（四则运算分配律和结合律）逻辑并行；分支选择的选路并行。

2) 计算功能硬件卸载的“数据—计算”抽象设计方法。

网络计算卸载到 FPGA 中，需要在 FPGA 的并行计算加速基础上，引入一种适合于流式计算 FPGA 的编程抽象，本文称其为“数据—计算”模型（Data-Computing, DC 抽象）。如图4-3所示，每个节点内都包括一个或一系列功能（函数 $F(x), G(x)$ 等），本文称其为“计算”，在不同的计算节点之间是数据流（以数据包为驱动），本文称其为“数据”。如何排列和组织这种“计算和数据”是 DC 抽象的核心内容。一般数字电路会

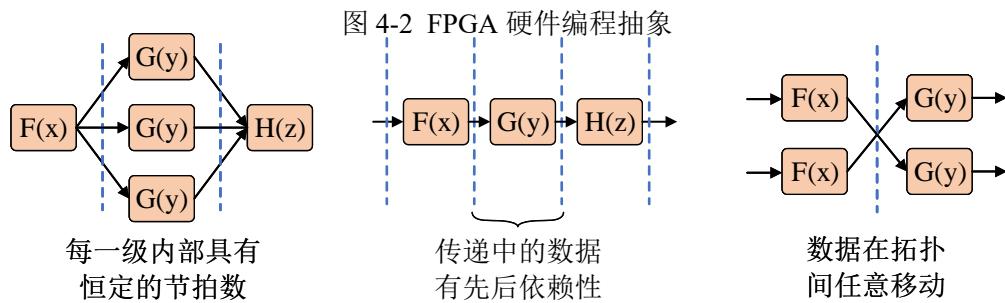
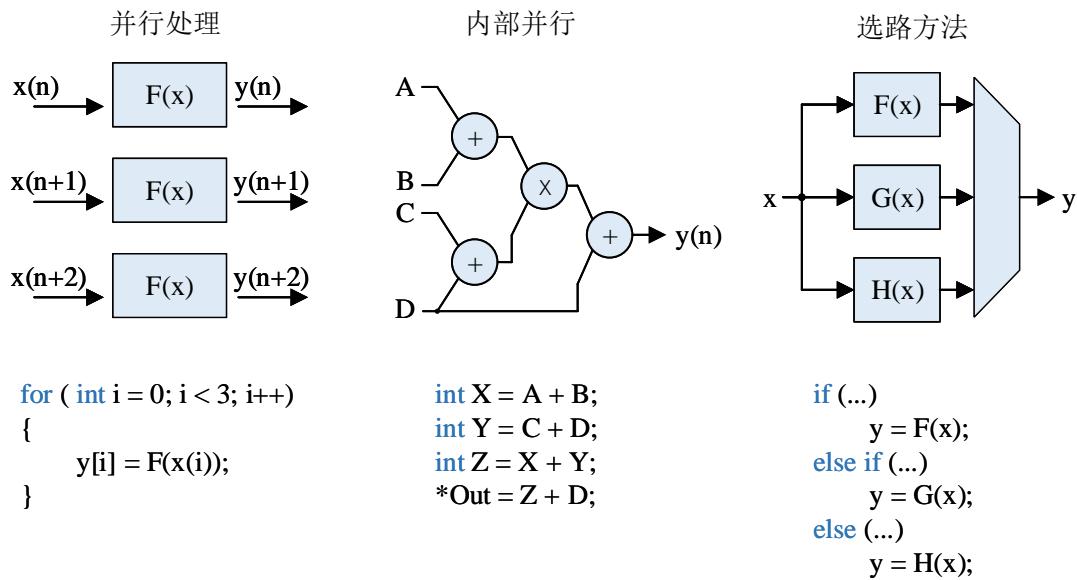


图 4-3 FPGA 加速流式计算的 DC 抽象方法与原则

设计为时序逻辑，也就是以时钟周期时间跨度为节拍的计算。各个逻辑块需要在同一个节拍内计算出相应结果。对于在数据处理时前后无关的计算可以归到内部并行中去，所以流式数据处理方法需要挑出数据计算时有前后相关的部分，并把它们按照一定拓扑串联起来。DC 抽象基于上述分析提出在 FPGA 内处理网络流式计算的加速的原则：

- 在流水线的每一级具有确定的节拍数。
- 数据前后内容具有依赖性。
- 有固定的流水线拓扑，但是数据在拓扑间可任意流动。

下面将重点介绍本文如何利用 DC 抽象原则，对基于可编程硬件的高时效性网络任务和高密度计算任务卸载加速的内容进行研究。

4.4 网络流量高性能数据通路

4.4.1 概述

如表4-1中所列，网络测量和分析是监控、调试和管理网络，进而设计反馈和优化网络结构以及改善网络服务质量的重要手段。近年来，互联网与网络应用的不断发展，网络内服务器遭遇的各类破坏与攻击也随之与日俱增，网络安全问题日益引起人们的重视。因此，在大型数据中心内，对入侵攻击的检测与防范、保障信息系统和网络系统

安全已经成为研究人员的共识。为了快速确定网络攻击的发生，如何设计一种可以实时检测网络流量的网关系统变得异常重要。例如，当网关系统发现大规模 DDoS 攻击时，可以主动做出丢包等拦截动作，但是对正常数据包应采取放行策略。

这类系统对低传输时延、高吞吐迫切需求。对于复杂网络协议和内容的分析，一般需要使用大型服务器的多核 CPU。由于网络中流的数目很多，可以将不同的流量分配到不同的并行 CPU 中去做并行入侵检测。但极高流速的网络数据包的捕获，分发等工作却面临很大的性能瓶颈。经过检测系统处理完的数据包还需按先后顺序汇聚，保证多核并行系统不会对原始流量数据包的内在逻辑产生影响。因其特殊的处理需求，以及数据包高吞吐特性，因此安全监测系统的网络收发部分是影响系统处理性能的重要瓶颈之一。本文的主要研究点为服务器端网络数据的接入、分发与回流，其余的内容检测等属于上层软件应用，不在本文讨论范围之内。

4.4.2 问题分析

对于流量接入而言，其挑战在于如何快速地捕获足够的网络流量。经典的如 TCP-Dump 以及 TCPTrace 等基于软件的流量捕获工具，可以完成将某网络接口数据包镜像并缓存的功能。简单来说数据包首先到网卡，然后通过软中断的形式触发内核对其处理，通过 DMA 写入内存，最后再被应用程序读取。软件首先使数据包经过 BPF（伯克利包过滤，BPF 能够通过比较第 2、3、4 层协议中各个数据字段值的方法对流量进行过滤）包过滤器，通过过滤的数据包放入 bufferQ 缓存（给应用程序读取数据包的缓冲队列），tcpdump 就是从 bufferQ 中抓取数据包。不难发现，这类操作必须由内核态到用户态两次传输，一个数据包需要经过多次 CPU 处理循环。

其主要问题有两点：操作系统时间精度低，软件处理丢包严重性能较差。首先，时间精度低。为了记录数据包接入的速度特征分布，需要为每个数据包记录到达时间戳，在数据包回放到网络时可最大限度地保证不影响每个流的原始特性。数据包到达后发送软中断，软件为其添加时间戳时已经有了系统误差。这种误差来自于操作系统时间管理与任务轮询的不精确性（例如图4-4^②）。这样的不精确会给流分布的记录造成误差，最终导致某一段数据流抖动（jitter）增大，改变数据流特性。加之在真实场景中，软件收发包通过虚拟层，协议栈，内核态，用户态等多步骤处理，累计的总体时间抖动很大。第二，性能低下。目前在 linux 服务器环境下的捕获的丢包率偏高（例如图4-5），与真实网络接口速度差距过大。由于需要对串行数据中每个数据包依次增加时间戳，因而在此过程中无法引入软件并行，从而仅依靠单核极易造成性能瓶颈。

流量回放技术基于流量捕获基础之上完成，除前文提到用于实时入侵系统外，在离线性能测试、模拟攻击等领域还有重要作用。人工构造的流量并不能完全重现真实环境中的网络，研究人员利用流量回放还能用以测试实时网络安全系统。

^② 本实验指在测试软件协议栈的系统时间精度，一种简单方式是通过 PING 协议的 RTT 时延来作为参考。如果软件处理稳定性很高，则每次定点 RTT 的数值应保持稳定。由于目前云服务商在系统内禁止网关的 PING 服务，本文通过协议栈与本地回环（127.0.0.1）之间的 RTT 来近似估计，以此计算出用户态到本地网关之间的抖动。但本文发现，即使测试本地网关，都依然有巨大的时间抖动性。

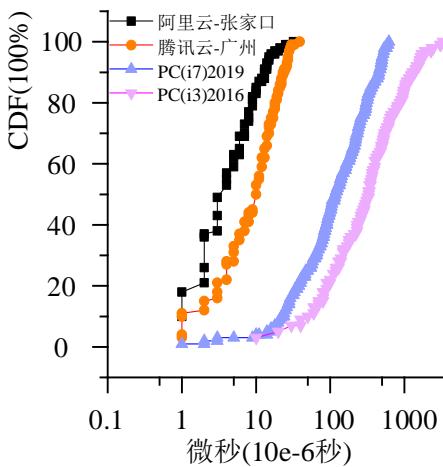


图 4-4 软件环境网络协议栈发包时间精度

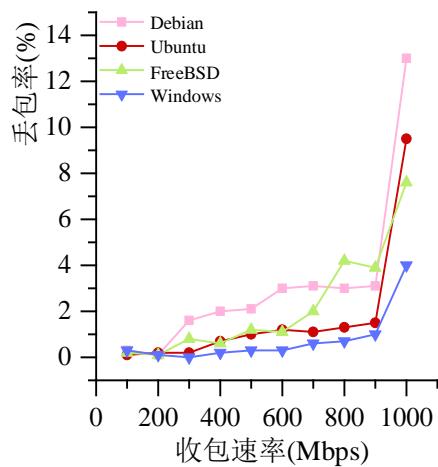


图 4-5 TCPdump 收包工具丢包率

为应对如今超大规模接口带宽，基于软件的流量捕获和回放系统遇到了极大的可扩展性问题，已无法满足目前的行业需求。本文以解决此问题为目标，在主机端使用可编程硬件网卡技术，来大幅度提升服务器捕获回放数据包性能。

4.4.3 设计

网络流量的捕获、统计与回放是统一的一套体系，在软件层面使用多核间的流水线完成加速。在流水线中的所有数据包，都可以看做同时受到各个前后功能的处理，这样可以大幅增加吞吐。本文的基本思路是将软件层的所有瓶颈功能全部下放到可编程硬件，只留轻负载的文件存储管理系统。当前基于软件的网络流量捕获系统收到数据包后由 CPU 处理所有操作（图4-6）。对此类型架构的一种典型优化方式为使用多核 CPU 并行加速。首先为每个 CPU 核心分配不同的任务，且应该尽量使任务负载均匀分配。之后数据流凭借共享内存的形式，依次串行通过所有功能核心。这样使多核 CPU 完成了类似流水线的结构。由于每一个核心不可能得到真正公平的任务负载，所以系统的最大处理能力受到链路内最窄处的制约。

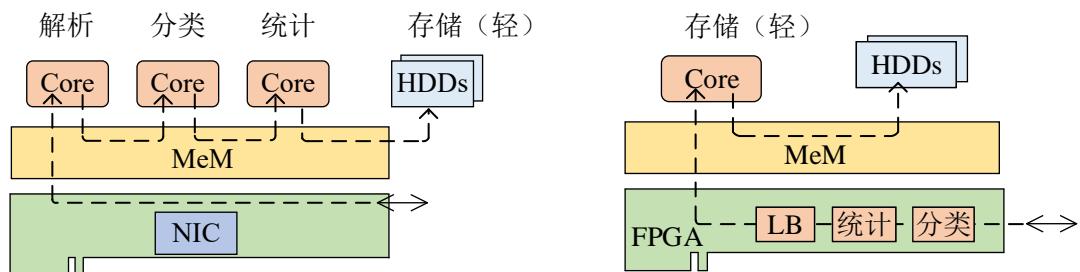


图 4-6 基于软件的设计思路

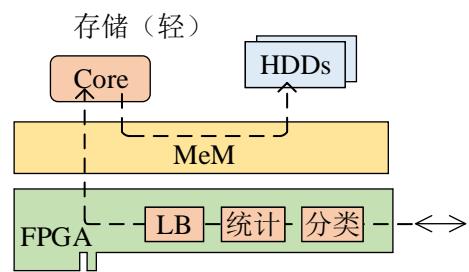


图 4-7 基于可编程智能网卡的设计思路

如图4-7所示为本文基于 FPGA 的硬件可编程智能网卡系统架构。本文将之前在软件中的大部分计算密集型任务全部下放到硬件。与软件系统相比，本文提出软硬一体化网络流量捕获回放系统有如下优点：

- 时间戳精度高。FPGA 运行主频为 125MHz，硬件逻辑可精确地在数据包到达时刻将系统时间插入数据包字段内。硬件系统使用相对时间，由于数据包接入与

回放只要保障相对顺序即可，时间戳无需使用操作系统内的绝对时间。125MHz 主频对应每一个时钟周期 8ns，因此系统对数据包的监测处理时间误差以 8ns 为单位，远远优于软件协议栈时间精度 ($10^3 - 10^6$ ns)。

- 时间同步性强。网卡中的相对时间计数器由一组寄存器构成。FPGA 内寄存器位置分布灵活，逻辑与寄存器组同步性好，这为数据包各个阶段的逻辑操作提供了统一的时间评价尺度。
- 流水线内无性能狭窄处。根据前文提到 DC 抽象方法，在数据计算密集型场景下，硬件系统内各个处理阶段应该有相同的节拍个数，因此保障了整个处理链路可预知的性能预期。
- 消除软件性能瓶颈。系统将密集型计算和控制卸载到硬件，在 CPU 中只保留了数据的存储管理功能。如果系统希望把捕获的数据暂存起来，待以后离线分析或者其他时间回放，则需要把大量数据包内容存储到计算机硬盘内。这部分操作相比之前的密集型计算负载更“轻”不是系统瓶颈，且易于并行扩展。它需要与高层文件系统、I/O 调度等交互，因此保留在软件层较为合适。

本文在流量捕获回放系统中间还加入了一种高存储优化的统计系统，此内容将在下一节单独介绍，不会影响本节内容安排。

4.4.4 电路实现以及协议

如图4-8所示为流量捕获回放系统 FPGA 内硬件核心逻辑架构，接下来分别介绍设计的核心思路。其中捕获通路中的统计电路采用了一种实时的带有数据压缩方式的设计，内容较为复杂。为了不影响论文结构层次，将其放在下一小节单独介绍。

1) 数据包硬件接口。

在智能网卡上，数据包首先经由物理网口 (phy_port) 进入板上电路，此时需要将网络通信的物理层信号协议转换为二层数据帧。一般由光电转换模块和物理层芯片结合处理。帧封装的数据包数据信息是串行高速总线协议，MAC 核逻辑将其转换为低速率的并行总线，并分离出数据的伴随时钟信号。之后为数据流（数据字）添加统一格式标记（控制字）。

2) 流量捕获—数据包处理通路。

在数据通路中，本文需要增添三个硬件模块：时间戳模块、流量均衡分类器和多路输出调度器。

- 时间戳模块。为每个到达的数据包添加实时的 64bit 时间，数据总线结构如图4-9所示。FPGA 数据通路主频为 125MHz，每个时钟周期时长 8ns。时间戳寄存器每个时钟周期都被触发一次自增 8 的加法操作。每个数据包到达有严格先后顺序，所以每个数据包的时间戳都是独一无二的。64bits 数据可以表达正整数范围达到 1.8×10^{19} 以上，从 ns 换算成年，相当于地球年的 500 年以上，数据位宽足够使用。
- 流量均衡分类器。流量捕获系统将数据包从网口传递到硬盘模组，整个链路中

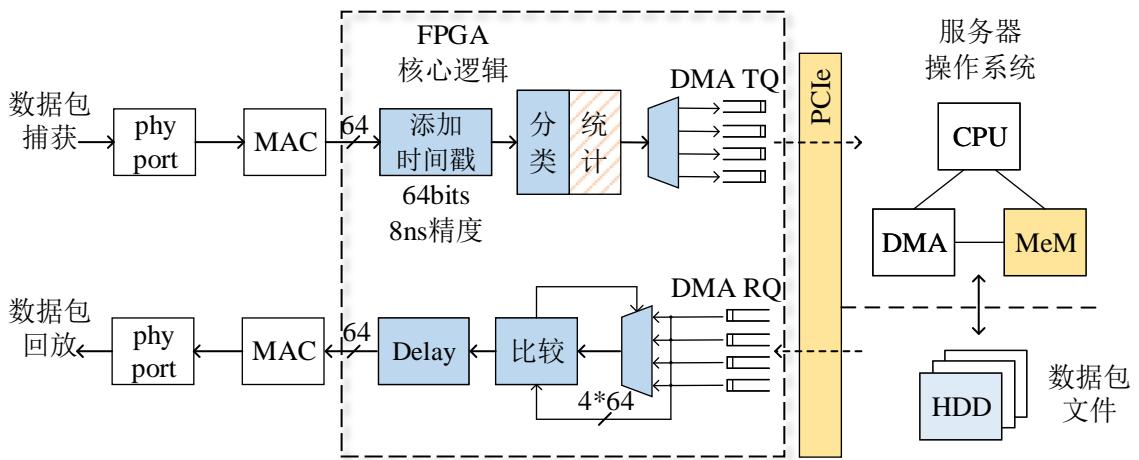


图 4-8 流量捕获与回放系统数据通路 FPGA 核心逻辑架构

最有可能成为瓶颈的地方是硬盘的 I/O 驱动，由于普通机械硬盘真实外部读写性能只有 1.5Gbps (SATA 接口) 左右^③，这极大地限制了系统的性能。为了增大系统的最终捕获速度，本文将大流量通过硬件哈希算法分类为 N 条子流量。子流量的流速期望就变为之前总流量的 $\frac{1}{N}$ 。哈希函数 (CRC-16) 的输入为网络流量数据包前五元组信息 (目的 IP 地址，源 IP 地址，目的端口号，源端口号，协议标签) 组成的 104bits 二进制数字。对于不包含五元组信息的数据包，可使用二层 MAC 地址 (96bits) 后补齐 0，来组成新的 104bits 数字。CRC-16 算法将输入数字转换成一组 16bits 的短数字，不同 hash 函数会转换出不同的值。但对于同一个 hash 函数不同的输入也可能得到相同的结果，最后我们选取结果的最后 2bits 位作为标记，以区分最终的缓存分支队列。一般的 CRC 函数都可以在同一个时钟周期内出结果，不会影响流水线性能。

- 多路输出调度器。多路调度器读取 hash 函数的输出结果并将此时的数据包按结果分发向不同队列。最终并行队列会分别上传到软件层，分别储存到不同的硬盘中。

控制字		数据字 64bits			
8'h FF		目的端口	长度	源端口	长度
8'h FE	时间戳				
:	数据包				
8'h 00	包结尾				

图 4-9 数据包格式与控制字标志

3) 流量回放—数据包处理通路。

流量回放是流表捕获的逆过程，数据从多列硬盘中读出并发送到 FPGA 内部多列

^③ 实验完成的板卡是以 PCIe2.0×8 为标准的主板插槽卡。互通总带宽为 $5 \text{ Gbps} \times 8 = 40 \text{ Gbps}$ ，因而 PCIe 总线不是系统瓶颈。

缓存中。FPGA 根据多队列数据流包头时间戳大小重新组合为一组数据流向板卡外发送，从而使系统可以完整模拟数据流现场状况。本文在回放通道中设计了两个硬件模块：时间比较模块、延迟开关。

- 时间比较模块。此模块读取所有队列中的时间戳，并返回时间最小的时间戳所在的队列编号。在软件中，查找 N 个数据中的最小值时间复杂度为 $O(N)$ 。利用硬件逻辑天然的并行优势，可以对 N 组数据两两比较得出结果后，再对结果两两比较，这样只需要 $O(\log N)$ 级比较即可输出最小值。值得注意的是，只有在比出最小值后取出队列首包，弹出后续包时间戳数据后，才能进行第二回合的比较。如果队列数过大，也许在同一个时钟周期内逻辑做不完比较而产生输出不稳定的“竞争与险象”，从而得到错误的结果。例如，输入队列有 64 条，比较器的层数有 $6 (\log_2 64)$ 层，从中找出最小值需要至少 2 个周期或者 3 个周期时间^④。如果查询时间超过一个时钟周期，数据在流水线中必然无法实现“背靠背”处理，总体处理流程都需要等待最耗时的这一级。
- 延迟开关。当选出最小时间戳的数据包后，并不是马上就可以发送出去，因为数据包与数据包之间有不固定的时间间隔。延迟开关的作用就是根据时间戳前后相对差值，让数据包等够所需的时间之后，再打开放行。这是提高数据包模拟真实环境相似程度的关键一步。

4.4.5 优化

本小结主要分析流量捕获与回放系统的设计性能，并且探讨系统向目前主流网卡单网口 100Gbps 演进的优化方案。

上节提到，系统 FPGA 的主频选用 125MHz，数据位宽 64bits，则数据通路的总处理性能为 $P = 8Gbps$ 。假设目标条件采集/回放速率 $T = 4Gbps$ ，系统的瓶颈处：硬盘 I/O 需要扩充 $\frac{4}{1.5} = 3$ 倍，为保证性能可留有 30% 的设计余地，则选取 4 块硬盘并行写入，总写入能力达到 6Gbps。回放通路的逻辑瓶颈在于时间戳比较模块，4 路数据可以在一个周期内比较完成，因而也满足处理性能大于实际需求。由于当时技术协议发展水平受限，设计板卡的需求与现在相比略低。

如要适配当前高速率环境下的新协议，本文需对设计瓶颈重新分析。FPGA 内主要数据通路性能必须要大于 100G（本文选取 200MHz 主频，512bits 数据位宽），需要网卡拥有 1 路 100G 的网口，PCIe3.0×16 ($8Gbps \times 16 = 104Gbps$) 的互通带宽。目前主流高性能固态硬盘的读写性能达到 10Gbps，因此我们需要选择 16 路分流并行存储，来解决存储 I/O 瓶颈，至此捕获通路无性能瓶颈。回放通路的性能瓶颈在于 16 个队列时间戳最小值比较，由于 16 队列需要至少 4 层并行 2 进 1 比较器，需要 2 个时钟周期才能完成。因此回放数据通路的处理性能只能达到 100Mpps，但是这并不意味，处理性能也会下降一半（到 50Gbps），只是在处理时无法满足最小包长的 100G 性能。如果通路内均为 128 字节包长，则系统也可达到 100G 线速。由于真实网络环境的平均包长度为

^④ 具体耗费时延与选用 FPGA 器件特性有关，例如更低 nm 制程响应速度越快，也与运行主频有关。

600 字节，留给每个数据包 10 个时钟周期的处理时间，因此在真实情况下性能是满足的。

4.5 基于硬件的高性能压缩统计方法

4.5.1 网络统计方法概述

随着云数据中心规模不断增长，基础设施架构自我设计制造成为主流，网络运营商因此有能力在网络监控和管理方面投入更多。目前许多基于云端和裸金属环境新提出的新应用（hypervisor，虚拟机管理）和算法（BBR，拥塞控制算法）对各种各样的网络测量结果产生极大的需求。网络测量是指在运行的网络中测量各种实时变动的状态参数，例如 RTT、包数量、流容量大小、流容量速度、拥塞程度等。这里大部分参数都可用被动方式进行测量，即，对运行中网络不引入任何影响。被动测量对网络运营商是一种有效的工具，例如，网络计费、安全监控、流量调度等。被动测量的过程一般分为 2 部分：首先，测量装置收集网络中的信息（通常以数据包驱动计数）；第二，软件代理定期收集测量结果，并汇报上层控制层。

为解决监测系统软件实现效率低下的问题，研究人员提出一类“高速—低速”混合存储器更新方案。首先将流维护在价格昂贵的高速缓存中（SRAM），此时 SRAM 为每条流的计数器开辟空间较小。当 SRAM 中某条流溢出时，再将此计数器值更新到 DDR 中，并为 SRAM 中对应的计数器清零。此方法有效减缓了 DRAM 的更新次数。但是会引入读值速度慢，硬件电路结构设计复杂等问题。此外，有工作提出将多个 DDR 用于并行统计以减小读值复杂度。然而随着数据量的增大，固定长度位宽的计数器依然有很高的溢出风险。

核心网络中，流量在转发节点总吞吐可达到上百 T，流量数目可达到上百万，而流量转发节点数量也成百上千。对于精细的流量统计任务，每秒钟的更新量可能会达到 150 亿次^⑤，而因统计所耗费的存储空间需求可以达到数百兆^⑥，软件代理不可能做到每包更新。因而大量的统计数据缓存在本地高速内存中。如此庞大的统计任务一般会由分布式设备进行加速以及分流统计，尤其在数据中心网络，服务器的数量众多，因此在流量的端处进行统计能够尽量减少计算存储和汇报的性能压力。

上一章提到，云计算网络的设计理念为尽可能降低服务器 CPU 因网络任务而带来的大量消耗。由于网络任务本身是以数据包为驱动的，在大型数据中心的服务器往往对外网络性能可达到 100G，数据包触发速度可达到 150Mpps。网络功能每增加一步操作都会引起数量庞大的额外计算。此外，统计功能占用的大量存储空间也成为设计挑战。CPU 中的一二三级缓存（cache）虽然速度极快，但容量相对较小（1MB 以内）。因此待更新值必然存储在大容量的外置存储器（DDR）中，DDR 的随机读写性能极低，面对如此高频率的更新需求，系统 I/O 带宽成为瓶颈。即使都将数据暂存于快速 cache 中，以目前主流服务器 CPU 的 2.5GHz 左右时钟频率估计，更新频率大概也只有主频

^⑤ 以 10Tbps 容量，转发 64 字节数据包计算。 ^⑥ 以 10M 条流量，每条流统计占用 8 字节宽度为例。

的十分之一或百分之一。两种因素互相钳制，导致使用主机 CPU 做细粒度统计开销极大，在 CPU 时间成本极大的云服务器领域，测量的效率问题成为亟待解决的一个难点。

4.5.2 问题分析

被动测量在网络中是一种基础服务，尤其在拥有可编程智能网卡的大型数据中心内，为实现高性能测量提供了可行性。目前 FGPA 板载高性能存储容量都有几十 MB，为实现各种应用提供了丰富的逻辑资源。然而如果将所有测量项目都实现在硬件中，依然会造成片上资源过度消耗。一方面，如果产生一种新的测量需求，就重新开发硬件逻辑，这会对网络操作人员造成严重的时间浪费。另一方面，因为 FPGA 其他逻辑资源也需要消耗一定数量的存储，各个逻辑模块需要均衡地使用存储资源。考虑到 FPGA 后期布局布线，板载资源使用率尽量小于全部资源的 70%。这为本文研究基于 FPGA 的测量技术带来了极大挑战。

本文基于一种在硬件中实现的可压缩每流测量算法，有效地解决了此问题，从而可以在硬件上为每一条流保持统计信息，并且只使用少量的存储资源。即使使用大存储容量的器件，如果在实际应用中不使之压缩，统计学上总有计数器值溢出的风险。

4.5.3 基于硬件设计压缩效果与问题

1) 压缩算法

目前基于抽样的压缩统计方法是一种新兴的测量设计思想，因为只须将一部分数据包被选中而送往统计模块，这样可以大大降低存储资源消耗，同时能够减少计数器累加次数。简单抽样对于均匀流量有很好的效果，但是对于各个流量大小不一的场景，小流的误差值会很高。现阶段各类抽样方法中可被证明拥有最佳估计误差的压缩算法是 DISCO^[105] 算法。DISCO 算法是一种可被证明的无偏估计压缩方法，DISCO 算法的核心目标是减少高速缓存内关于每条流计数器的位宽占用大小，并且对包个数以及字节数的压缩都可适用。

假设 c 是计数器中存储的值（被压缩后的）， n 为统计流量大小的真实值，令函数 $n = f(c)$ 表示二者的压缩数量关系，

$$f(c) = \frac{b^c - 1}{b - 1} \quad (4-1)$$

式中， $b > 1$ ，且 b 为系统预先设置的固定值。可发现， $c = f^{-1}(n)$ 为一个凸函数，统计数值 c 的增长趋势一定是亚线性的。随着 c 增大， $f(c)$ 可近似为一个指数函数。因此对于一个足够大的 n ，函数 $f(\cdot)$ 可将原来线性增加的数值压缩至 $O(\log n)$ 。每当长度为 l 的数据包到达系统，针对此数据包的计数器应增加 $\Delta(c, l)$ ，此时 c 为当前计数器压缩后的值。根据定义 $f(c)$ ，可以得到 $\Delta(c, l) = f^{-1}(l + f(c)) - c$ 。如果存储 c 的计数器采用浮点数计数，则每次对 c 算出新增 $\Delta(c, l)$ 并相加后，得到的 c 都可以无误差地表达 n 的真实值。但存储带小数位的浮点数会占用更多存储空间，为了提高存储效率，需假定 c 是一个正整数。

```

1  $v = \text{random}(0, 1)$ 
2 分别计算  $\delta(c, l)$  和  $p_d(c, l)$ 
3 if  $v \leq p_d(c, l)$  then
4    $c = c + \delta(c, l) + 1$       // 显然, 当  $p_d$  越大,  $c$  最终向上取整的概率也越大
5 else
6    $c = c + \delta(c, l)$           // 向下取整

```

算法 4-1 DISCO 算法计数器估计更新方法

DISCO 算法为努力将存储空间降低引入了一种概率估计。由算法4-1, 系统可以获取整数估计后的压缩值 c 。其中 $\delta(c, l)$ 和 $p_d(c, l)$ 由下列公式定义:

$$\delta(c, l) = \lceil f^{-1}(l + f(c)) - c \rceil - 1 \quad (4-2)$$

$$p_d(c, l) = \frac{l + f(c) - f(c + \delta(c, l))}{f(c + \delta(c, l) + 1) - f(c + \delta(c, l))} \quad (4-3)$$

经一段时间的统计, 监测设备向控制器上报计数值 c , 控制器可由公式4-1快速反向推导而出, 同时此估计方法的无偏性由论文^[105]给出。

2) 压缩算法硬件实现的效果

算法4-1中, 除加减法与比较、判断之外, 求函数 $f(\cdot)$ 与 $f^{-1}(\cdot)$ 的关键计算指令是求“指数”与“对数”。计算机快速计算指数, 一般使用自然底数变换法, 辅以查询算数表以及对精度的层层消除, 计算时间由原来硬解法的 $O(n)$, 降低至 20 个指令周期左右。对数函数可以使用泰勒级数展开来近似计算, 展开后计算方法均为指数乘法和加法, 进而再次转化为指数复杂度的计算。

DISCO 算法提出后, 由软件实现并达到单 CPU 核心 11Gbps 处理能力。目前的服务器终端网络通信容量已由 40Gbps 向 100Gbps 发展, 继续扩展 CPU 核数满足处理需求已显低效。网络通信包数据量巨大, 即使使用快速算法也无法满足网卡的线速处理需求。100G 网卡的数据包处理速度需求为大约 150Mpps, 数据通路内时钟主频不会超过 300MHz。按上限计算, 每个数据包的处理时间余量只有 2 个时钟周期 (与快速算法相比, 本文需求超过其能力 1 到 2 个数量级)。当然流水线设计方法可以把计算过程拉长到多节拍, 从而保证吞吐量。但需要注意的是本算法中数值 c 有先后依赖关系, 即: 首包处理时, c 若没有完成更新, 第二个数据包无法进入处理通道。因此从全局看, 更新环节无法真正打开为多级流水线。

注意到公式中所有的底数 b 为预先确定值, 因此在实践中可以存储只关于 b 的计算表, 这样可以最大限度减小计算复杂度, 也比较节约计算用存储资源。假定计算表的深度为 3072, 宽度为 32bits, 选定 $b = 1.006$ 。系统支持的最大计数数量等效于 $f(3072) =$

$1.59 \times 10^{10} Bytes = 15GB$ 。而计数器 c 只需要 12 位（保存压缩后的值， $2^{12} = 4096$ ），而在不压缩的线性计数下则需要至少 34bits 的存储位宽 ($2^{34} = 1.71 \times 10^{10} > 1.59 \times 10^{10}$)，即：对于指数计算需求 $b^X (X \leq 3072)$ ，系统可以通过单周期查表方法获取结果。

使用查表法加速指数计算可获得很好的效果，但对数计算 ($\log X$) 即指数反函数计算无法实现一周期出结果。对数计算一般反向利用指数表进行二分估计。如图4-10所示，左边为上述的指数预算算查找表。初始化值， $c=0$, $f(c)=0$ ，假设此时到来数据包长度为 1500 字节。根据公式4-2计算 $\delta(c, l)$ ，以及公式4-3计算 $p_d(c, l)$ 。首先，在计算 $\delta(c, l)$ 时，最关键需要计算 $f^{-1}(l + f(c))$ ，带入 $l = 1500$ 、 $f(0) = 0$ ，得到：

$$f^{-1}(1500 + 0) = x \rightarrow f(x) = 1500 = \frac{b^x - 1}{b - 1} \quad (4-4)$$

带入 $b=1.006$ ，经过变换：

$$b^x = 1500 \times (b - 1) + 1 = 1500 \times 0.006 + 1 = 10 \quad (4-5)$$

因此，计算 $f^{-1}(1500 + 0)$ 的结果可以等价为：在指数计算表中查找当 x 等于多少时指数值为 10。由于指数值是非连续、非等差数列，查找方法只能使用二分法逼近。如图4-10右侧所示，在 3072 个表项中从第一行开始进行二分查找至少需要 12 次跳转。

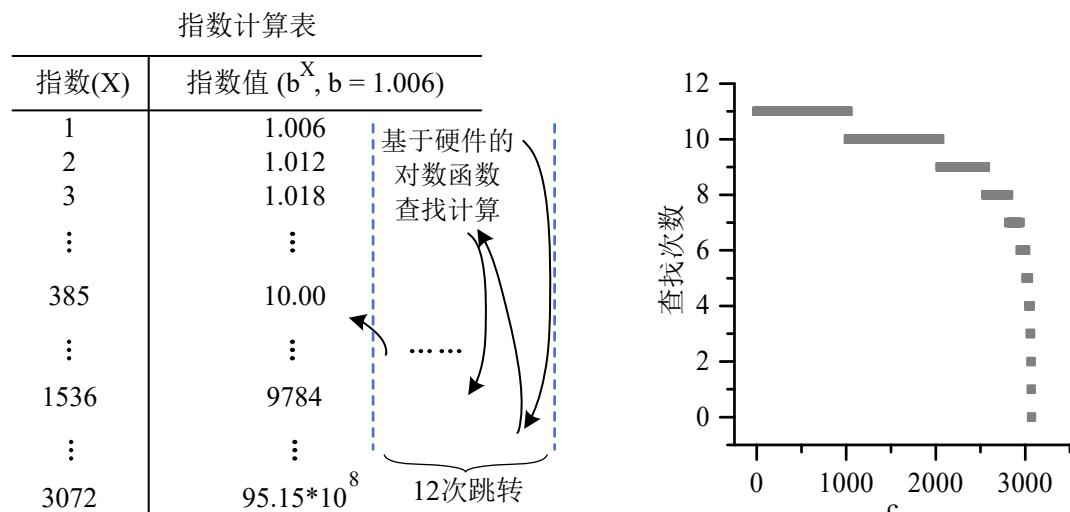


图 4-10 反向利用指数表进行二分估计以求得对数函数值

图 4-11 当流量中包长 $l = 1000$ 时利用指数表求对数函数值的查找次数

计算出 $\delta(c, l)$ 后， $p_d(c, l)$ 可由指数查表与 $\delta(c, l)$ 结果共同计算。本文发现，由于不同的 c 数值与不同 l 数值，基于硬件计算 $\delta(c, l)$ 的时钟周期并不固定，例如当前 $c=1536$ ，那么对数查找的起始位置便不在第一行，而在整体表的中间位置，这样搜索空间会缩小，因此查找次数也会变少。如图4-11所示，假设流每个数据包大小统一为 $l = 1000$ ，当 c 的值越来越大，查找次数会往减小的趋势发展。

3) 存在问题的分析

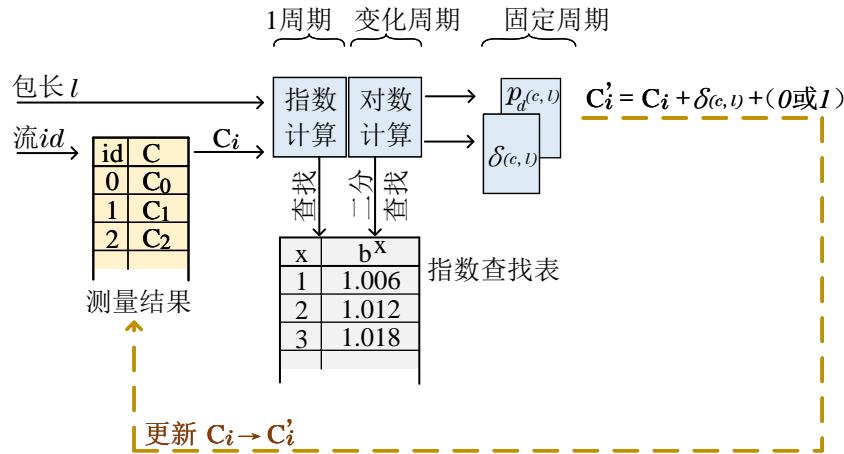


图 4-12 压缩统计硬件流水线设计 (未优化)

本文总结上述分析，并将基于软件的 DISCO 算法硬件流水线在图4-12中展示。当数据包到达系统，首先获取当前对应流的统计量 C_i ，之后由指数计算和对数计算模块求出中间变量，最后通过四则运算得到关键判定值 $\delta(c, l)$ 和 $p_d(c, l)$ 。对其进行性能分析：硬件设计为提高吞吐量一般采用流水线设计。为综合前后分级的节拍，要求每一级流水线的时延固定。本文发现二分查找虽然平均时延只有 7 周期，但最长时延可达到 12 周期，最短只需 1 周期，因此为了使流水线对外表现稳定，需要在硬件设计逻辑是增加许多配合打拍的逻辑，资源效率较低。另外在计算关键判定值时需要使用两次除法器。逻辑除法在 FPGA 设计中一般耗时需要 40 个周期以上，虽然除法器可以以流水线方式工作以提高吞吐，但本算法有一个更新闭环，在只有一条流量的时候，前后两次更新时间内不可进入流水线计算，因此一次更新所耗费的时长大约为 50 个周期。假设 FPGA 数据通路主频是 300MHz，因而更新频率只有 6Mpps。若对应 64 字节最小包，硬件加速后也只能达到 4Gbps 线速，与本文 100Gbps 设计目标相差甚远。

4.5.4 优化

1) 符合 DC 抽象的算法优化方法

前文提到的优化设计方法原则 1：每一级应有固定的节拍数。由于更新压缩表有前后强依赖，因此找到如何缩短整体处理时间，以及将可变动节拍计算逻辑固定化（应向减少节拍的方向优化）是本文的核心贡献。

本文发现，若 c 值处于比较大的阶段，当进行每包更新时， c 与更新值 c' 的差值最大为 1，即：公式4-2中 $\delta(c, l) = 0$ 。因此，根据算法4-1，有：

$$c' = c + \delta(c, l) + 1 = c + 1 \quad (4-6)$$

这是一个很重要的现象，前一节提到，若想加速本环节内的计算速度，须减少更新循环内耗费的时钟周期数。要将时序可变的逻辑修正为固定逻辑，且应尽量减小。如

果只需要判断 c 更新后的值加一与否，也许可以避免掉大量的对数二分查找过程。另外对于当 c 比较小的时候，本文将其计数转化为线性增加，因此计算复杂度也很低，若 c 较小也表明占用的计数位宽度不会很大。接下来本文分析此方案的可行性。本文首先分析，当 $\delta(c, l) = 0$ 时系统状态。由公式4-2与4-3，本文得到：

$$\begin{aligned}\delta(c, l) &= \lceil f^{-1}(l + f(c)) - c \rceil - 1 = 0 \\ \lceil f^{-1}(l + f(c)) - c \rceil &= 1\end{aligned}\quad (4-7)$$

因此，得到：

$$\begin{aligned}0 &\leq f^{-1}(l + f(c)) - c \leq 1 \\ 0 &\leq f^{-1}(l + f(c)) - f^{-1}(f(c)) \leq 1\end{aligned}\quad (4-8)$$

由于 $f^{-1}(\cdot)$ 是增函数，且 $l > 0$ ，显然上述公式为正值，因此上述公式可变为：

$$\begin{aligned}f^{-1}(l + f(c)) - f^{-1}(f(c)) &\leq 1 \\ f^{-1}(l + f(c)) - c &\leq 1 \\ f^{-1}(l + f(c)) &\leq c + 1 \\ l + f(c) &\leq f(c + 1) \\ f(c + 1) - f(c) &\geq l\end{aligned}\quad (4-9)$$

在以太网规范中，包长 $l \in [64, 1518]$ ，根据上述不等式可知：须满足左边大于 1518 则不等式恒成立：

$$f(c + 1) - f(c) \geq 1518 \quad (4-10)$$

由于 $f(\cdot)$ 是增函数且导数始终大于零，则一定可找到一个最小的 c_{lmin} 值，满足：对于任意 $c \geq c_{lmin}$ 时，均有上式 (4-10) 成立，也即 $\delta(c, l) = 0$ 。

下面关注此条件下 $p_d(c, l)$ 的计算过程，DISCO 算法4-1对 $p_d(c, l)$ 的描述为：系统对压缩计数器以概率 $p = p_d(c, l)$ 增加 $\delta(c, l) + 1$ ，以概率 $p = 1 - p_d(c, l)$ 增加 $\delta(c, l)$ 。由上文，因 $\delta(c, l) = 0$ ，则此阶段的更新算法为：系统对压缩计数器以概率 $p = p_d(c, l)$ 增加 1，以概率 $p = 1 - p_d(c, l)$ 增加 0。

由公式4-3、以及条件 $\delta(c, l) = 0$ ，得到：

$$p_d(c, l) = \frac{l + f(c) - f(c + \delta(c, l))}{f(c + \delta(c, l) + 1) - f(c + \delta(c, l))}$$

$$\begin{aligned}
 &= \frac{l + f(c) - f(c+0)}{f(c+0+1) - f(c+0)} \\
 &= \frac{l}{f(c+1) - f(c)}
 \end{aligned} \tag{4-11}$$

观察上式，本文发现，当 $c \geq c_{lmin}$ ，DISCO 的计算操作可化简为算法4-2：

Input: 当前统计值 c

Output: 当前统计值的更新值 c

```

1  $v = random(0, 1)$ 
2  $p_d(c, l) = \frac{l}{f(c+1) - f(c)}$  // 计算概率值
3 if  $v \leq p_d(c, l)$  then
4    $c = c + 1$  // 根据概率值，计数器只增加 1
5 else
6    $c = c$ 

```

算法 4-2 当 $c \geq c_{lmin}$ 时约化的压缩算法

Input: b - 函数 $f(c)$ 中的预定义参数

Output: c_{lmax} - 基于硬件 DISCO 压缩算法中判断标志位

```

1  $PKT\_Length\_MAX = 1518$ 
2  $c = 0$ 
3 while True do
4    $c++$ 
5    $f(c+1) = \frac{b^{c+1} - 1}{b - 1}$ 
6    $f(c) = \frac{b^c - 1}{b - 1}$ 
7   if  $f(c+1) - f(c) \geq PKT\_Length\_MAX$  then
8      $c_{lmax} = c$ 
9   return  $c_{lmax}$ 

```

算法 4-3 求解硬件压缩算法不同阶段判定标志位 c_{lmax}

但此式中依然包含了硬件除法，由前文分析知，此步骤须耗费数十个时钟周期^⑦，因此本文将概率计算公式继续化简为查表算法。这将解决本文提出优化希望的第二个重点问题：计算依赖数据之间的时钟周期数量过大。

观察公式4-11，简化后概率 $p_d(\cdot)$ 与两个变量 c, l 有关，且计算时只依赖于简单函数计算过程。 c 的范围在 $[1, 3072]$ 之间， l 的范围在 $[64, 1518]$ 之间。因此 c 与 1 合成的总

^⑦ 根据不同结果保留精度以及除数被除数位宽需求，商用 Xilinx 除法器 IP 核须要耗费 40 到 80 个时钟周期不等。

体表项有 4 百万个，总共需耗费片上存储空间约 32Mb–128Mb^⑧。由于 FPGA 内高速片上存储资源相对较少，实验用 ONetCard 网卡只有 16Mb，然而目前比较高端的 FPGA 片上存储也只有 100Mb 左右^[156]。设计过程中需要折中考虑资源占用与计算速度的互相矛盾的关系。显然查表占据了过多硬件资源，在工程实践上是不合理的。

接下来继续观察公式4-11，可发现：

$$\begin{aligned} p_d(c, l) &= \frac{l}{f(c+1) - f(c)} \\ &= \frac{1}{f(c+1) - f(c)} \times l \end{aligned} \quad (4-12)$$

上式表明，参数 $p_d(c, l)$ 可分解为两部分，第一部分只与 c 相关，另外一部分是一个乘数系数 l 。因此，本文将查表过程分解为两步：第一步，根据参数 c 查找数值 $p_c = \frac{1}{f(c+1) - f(c)}$ ，本文称此查找表为 PLUT (p_d Look Up Table)；第二步，计算 $p_d = p_c \times l$ 从而得到所需概率 $p_d(c, l)$ 。

Input: b - 函数 $f(c)$ 中的预定义参数; c_{lmax}

Output: PLUT - 快速概率计算查找表

```

1 Depth = 3072                                // 查找表最深为 3K 条表项
2 PLUT[3072]
3 for c = clmax, clmax + 1, ..., Depth do
4   f(c + 1) =  $\frac{b^{c+1} - 1}{b - 1}$ 
5   f(c) =  $\frac{b^c - 1}{b - 1}$ 
6   pc =  $\frac{1}{f(c + 1) - f(c)} \times 2^{21.4}$ 
7   PLUT[c] = pc                            // 保存表项内容

```

算法 4-4 求解快速概率计算 PLUT 查找表

系统在初始化之前需计算 PLUT 表项中的值。概率值是一个归一化的数值，但在硬件中浮点数的处理比较复杂且存储效率低。本文将概率归一化为 32bits 最大的整数值 ($2^{32} = 4294967295$)，因此 $p_c \times l$ 不应超过 2^{32} ， p_c 的最大值应满足 $l_{max} \times p_{cmax} = 2^{32}$ ，得到：

$$p_{cmax} \leq \frac{2^{32}}{1518} = 2829359.2 = 2^{21.4} \quad (4-13)$$

因此硬件中除了一个深度为 3k 的 PLUT 表，还需要一个最大 22bits×12bits (PLUT× l) 的窄单周期乘法器^[157] 即可，最大总共消耗两周期即可完成所有计算。设计中的查

^⑧ 以 8bits 或 32bits 为结果存储位宽计算。

找表占片上存储容量约 9KB (=3K 深度 × 3Bytes)，远小于板上资源限制条件。

本文给出对于任意的系统测量需求范围，计算上文提到的 c_{lmin} （算法4-3）以及 PLUT 表（算法4-4）的计算方法。这两组参数均在系统初始化之前由控制器离线计算完成，在系统运行时加载如硬件查表空间即可。

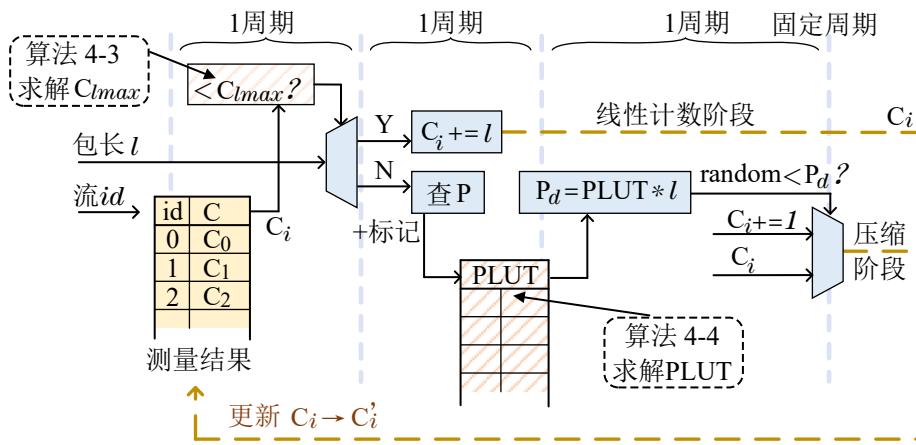


图 4-13 压缩统计硬件流水线设计（优化）

图4-13展示了优化后的压缩计数架构设计（即算法4-5所描述的快速压缩更新的完整过程）。Magic_Number 值由 c_{lmax} 带入函数 $f(c)$ 中得出，第一阶段，系统采用线性增长方式，系统判断当线性增长的统计量大于 Magic_Number 后，统计方式改为压缩方法（第二阶段）。为快速区分某统计值所处统计阶段，本文将统计量 C 的最高位设置为标记位，在第一阶段最高位为“0”，第二阶段时标记为“1”。

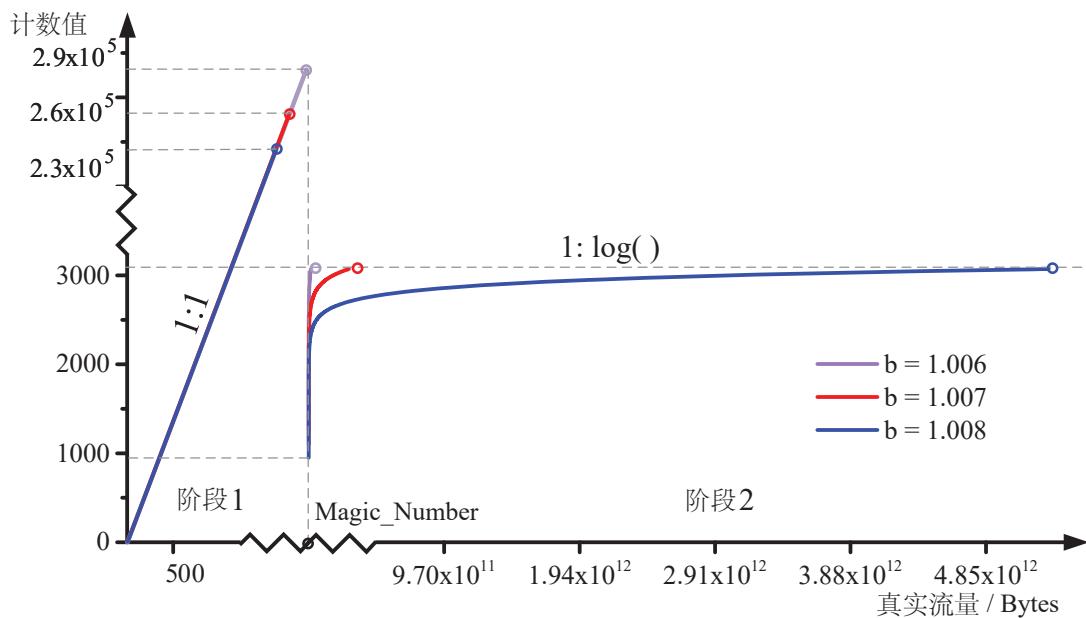


图 4-14 不同参数 b 下的计数曲线

本文提出的新压缩方法具有极大灵活性，在运行时用户可根据自身对于统计精度、统计容量大小 (b)，资源消耗率 (Depth)，来自行选择计算查找表内容。如图4-14，展示

Input: l - 每个到达数据包的待统计包长度; id - 每个数据包的流分类标号;
 $Counter_Table[]$ - 统计计数器存储组;
 c_{lmax} - 基于硬件 DISCO 压缩算法中判断标志位

Output: 完成快速更新统计表 $Counter_Table[id]$

```

1  $f(c) = \frac{b^c - 1}{b - 1}$                                 // 初始化过程
2  $M\_Number = f(c_{lmax})$                           // 计算判断标志位所对应的统计容量
3  $Counter\_Table[id] = 0$                            // 计数器初始值均为 0
                                         // 循环开始, 针对每个到达数据包都做计算
4 for Each incoming packet do
5    $v = random(0, 2^{32})$     // 获取一个随机数字, 范围为 0-4 字节内任意数字
6    $c_{current} = Counter\_Table[id]$ 
7   if  $c_{current} + l \leq M\_Number$  then
8      $c' = c_{current} + l$       // 判断是否属于第一阶段, 只进行线性统计并更新
9      $Counter\_Table[id] = c'$ 
10  else
11    if first time get into 2nd stage then
12       $l = l - (M\_Number - c_{current})$  // 首次进入第二阶段须在存储数据的
13      最左 bit 位增加标志位, 并初始化
14       $c_{current} = c_{current} | 22'b10_0000_0000_0000_0000$ 
15       $c_{current} = c_{current} \& 22'b10_0000_0000_0000_0000$ 
16       $p_c = PLUT[c_{lmax}] \times l$ 
17      if  $v \leq p_c$  then
18         $c' = c_{lmax} + 1$ 
19         $c_{current} = c_{current} \& c'$ 
20      else
21         $c_{current} = c_{current} \& c_{lmax}$ 
22       $Counter\_Table[id] = c_{current}$           // 第二阶段进行压缩统计并更新
23  else
24     $c = c_{current} \& 22'b01_1111_1111_1111_1111_1111$ 
25     $p_c = PLUT[c] \times l$ 
26    if  $v \leq p_c$  then
27       $c_{current} = c_{current} + 1$       // 第二阶段为压缩计数阶段并进行更新
       $Counter\_Table[id] = c_{current}$ 

```

算法 4-5 硬件统计压缩算法（优化）

了在不同计数容量需求下，修改参数 b 来扩展计数器统计容量范围（从 1.5×10^{10} Bytes 到 5.3×10^{12} Bytes，扩容 353 倍）。此时计数器位宽（20bits）与查找表 PLUT 的位宽深度（24bits，3072 项）均保持不变。

例：当统计量大于 294495 后进入第二阶段，此时 C 存储压缩后的数值。

输入：C=1254
输出：PLUT=RAM(1254) = 2371876

$$\begin{aligned} p(c,l) &= \text{PLUT} * \text{pkt_length} \\ &= 2371876 * 512 \\ &= 1214400512 \end{aligned}$$

$b=1.006$	RAM
$M_Number = f(1250)$	
= 294495	
Addr	PLUT
0	-
1	-
:	:
1249	-
1250	2429315
1251	2414826
1252	2400424
1253	2386107
1254	2371876
:	:
3070	45
3071	45

图 4-15 $b=1.006$ 时的 PLUT 查找表样例

图 4-15 展示了当 b 取 1.006，深度 3072 时，概率查找表（PLUT）存储内容的样例。若线性统计量超过 294495 后，进入第二阶段进行压缩统计。当此时 C 存储压缩值为 1254，且新入包长为 512 字节，则经过查表与计算得到 $p_c = RAM(1254) \times pkt_length = 2371876 \times 512 = 1214400512$ ，与随机数生成器比较后判断是否 +1，即可获得新的计数值。最终本文将耗时超过 50 个时钟周期的压缩计算过程缩短到 3 个时钟周期内完成。

2) 无偏估计证明

整理本文提出的计数器更新规则：

$$f(C) = \begin{cases} C & C \leq Magic_Number \\ \frac{b^C - 1}{b - 1} & C \geq C_{lmax} \end{cases} \quad (4-14)$$

其中 C 为计数器中存储值， b 为固定常数。定义阶段转换时系统状态：

$$\begin{aligned} f(C_{magic}) &= C_{magic} \\ &= \frac{b^{C_{lmax}} - 1}{b - 1} \quad (C_{magic} = Magic_Number) \end{aligned} \quad (4-15)$$

本文定义统计过程：统计从第一阶段开始线性累加，当统计量恰好等于 $Magic_Number$ 时，转换为第二阶段，其中判断数值 $Magic_Number$ 定义。 C_{lmax} 由（算法 4-4）由（式 4-15）定义，且此时压缩后的存储值（ C_{lmax} 为压缩后计数的初始值）所表达的实际值由

规则 $f(C)$ (公式4-14) 定义, 且初始值所表达真实值与 $Magic_Number$ 相等。

本文将证明上述的统计估计函数 $f(c)$ (c 为计数值) 在累积统计数据包长度 l 的各个阶段均为无偏估计。首先定义数据包统计过程, 不失偏颇, 本文假定针对某一条流进行叙述。计数过程的每次累加由数据包触发, 假设所有阶段内共有 m ($m \in N^*$) 个数据包到达并触发统计过程。数据包长度分别为 l_1, l_2, \dots, l_m ($l_i, i \in N^*$)。 c_i 为第 i 个数据包触发系统更新后的计数值。

定理 4.1: c 为公式4-14中的统计值, 其中 $f(c)$ 为真实统计量的估计值, 则 $f(c)$ 是一种无偏估计。即, 对任意 $i \in N^*$, 有:

$$E[f(c_i)] = \sum_{i=1}^m l_i \quad (4-16) \quad \square$$

证明: 首先, 考虑函数 $f(c)$ 的第一阶段。由于线性计数, 计数值等于累加值, 存在特殊情况 n ($n \leq m$), 使得:

$$c_i = \sum_{i=1}^k l_i \quad k \leq n; k, n, m \in N^* \quad (4-17)$$

又根据定义 (公式4-14), 可得:

$$\begin{aligned} f(c_i) &= c_i \\ &= \sum_{i=1}^k l_i \quad k \leq n \end{aligned} \quad (4-18)$$

因此:

$$\begin{aligned} E[f(c_i)] &= E(c_i) \\ &= c_i \\ &= \sum_{i=1}^k l_i \quad i \leq n \end{aligned} \quad (4-19)$$

其次, 考虑第二阶段, 前文工作^[105] 已经证明:

$$E\left[\frac{b^c - 1}{b - 1}\right] = \sum_{i=1}^m l_i \quad i \leq m \quad (4-20)$$

令 p 为第二阶段内的某一个数据包, 即 $n \leq p \leq m$, 根据定义4-15可得:

$$f(c_n) = c_n = \frac{b^{c_n} - 1}{b - 1} \quad p = n \quad (4-21)$$

由4-20与4-21得到:

$$E[f(c_p)] = E\left[\frac{b^{c_p} - 1}{b - 1}\right] = \sum_{i=1}^p l_i \quad p = n \quad (4-22)$$

由于 $p=n$ 时无偏性成立, 因此当 $p>n$ 时, 可直接应用结论4-20得证无偏性:

$$E[f(c_p)] = \sum_{i=1}^p l_i \quad n < p \leq m \quad (4-23)$$

联系4-19、4-22与4-23得到:

$$E[f(c_p)] == \sum_{i=1}^p l_i \quad p \in [1, m] \quad (4-24)$$

以上证明假定数据包 $1, 2, \dots, n$ 恰好组成了第一阶段, 且数据包 $n, n+1, p, \dots, m$ 恰好组成第二阶段, 并得证。另需证明一般情况, Magic_Number 无法由确定数量包长度之和组成, 即:

$$\sum_{i=1}^n l_i < C_{magic} < \sum_{i=1}^{n+1} l_i \quad (4-25)$$

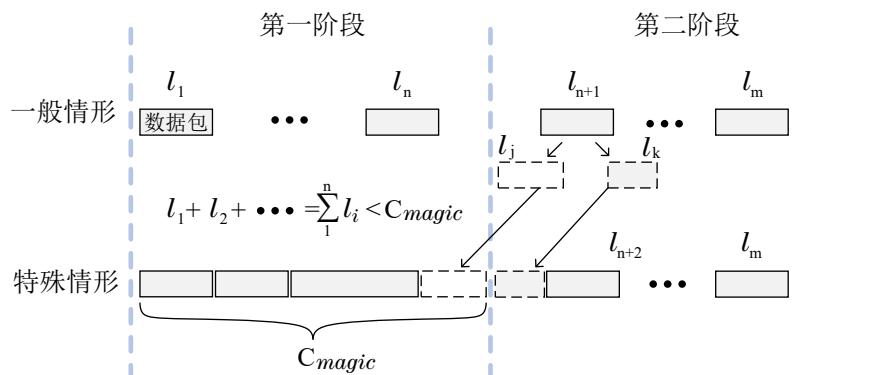


图 4-16 一般情形转化为特殊情形得以证明

对此 (根据算法4-5), 如图4-16, 将第 $n+1$ 个数据包的包长度 l_{n+1} 分为两部分: $l_{n+1} = j + k$, 得到两个序列 $n' = 1, 2, \dots, n, j$, 以及 $p' = k, n+1, n+2, \dots, m$ 且一定有 j 使得:

$$\begin{cases} E[f(c_{n'})] = \sum_{i=1}^{n'} l_i = \sum_{i=1}^n l_i + j = C_{magic} & n' \leq j \\ E[f(c_{p'})] = \sum_{i=1}^{p'} l_i & j \leq p' \leq m \end{cases} \quad (4-26)$$

因此，归约到特殊情况的结论（4-24）。证毕。 ■

4.6 软硬一体化的系统实验平台

基于软件的网络实现平台，例如 OVS^[80], cPqd^[158] 软件交换机，易于修改与部署，且可以配合网络拓扑仿真框架进行流量测试。但其仿真性能受限于 CPU 架构，交换性能和仿真时间随着流量和节点的增加而快速变差。这种通常的实验设施无法完成本文对于高精度高性能设计的验证。因此对于本章节介绍的两个关键内容，本文使用基于硬件的真实网卡进行流量测试。本文实现了一款带有 PCIe Gen2 × 8 主机接口的 FPGA 的智能网卡（ONetCard），配合相应的软件驱动，实现并打通了数据通路。

4.6.1 软件

实验平台包括两部分，一部分是主机运行的软件驱动程序，另一部分是 FPGA 网卡中运行的逻辑编码。数据帧从硬件传输到软件依靠 CPU 中断的通知，所以当网络中的数据包传输速率非常快的时候，单位时间产生中断次数也非常多，数据包的处理过程被大量中断干扰，从而降低了性能。目前数据从硬件到软件的传输方式通常为 DMA，DMA 可以减小搬运数据所消耗的 CPU 时间，从而提升传输效率。针对网络流量捕获和回放的 DMA 驱动软件模块，通过 PCIe 接口在主机内存与网卡内收发 FIFO 直接传递数据。要求高速传输，因此 DMA 设计的时候要考虑“高带宽，低时延，低中断频率”。本模块大体分为三个主要部分：PCIe，DMA，数据包队列，各个模块功能特性如下：

- PCIe 模块。PCIe 接口为 Gen2 × 4，总带宽为 $4 \times 5\text{Gbps} = 20\text{Gbps}$ 。PCIe 接口 IP 采用 Xilinx 提供的 7Series_IntBlock_PCIE IP core，总线宽度为 128bits，总线时钟频率为 125MHz。
- DMA 模块。使用基于描述符换的 Scatter-Gather 型 DMA。使用多通道 DMA，通道数量可配置。低中断频率，在系统内存中设置多块缓存区采用“乒乓”操作（或基于简单的容量计数以及计时机制）控制中断频率的速度。包计数中断域值（或缓存块大小）计时器均可配置。
- 数据包队列。软件中数据包存储队列数可配置。收发队列互相独立，互不干扰。

1) 从网卡到内存。

主机为每一个收包队列建立一个环形 DMA 描述符列表，假定包括 10 个描述符，这 10 个描述符连续存放。每个描述符信息包括：有效数据包数量，对应缓存起始地址，对应缓存总占用容量。完成之后主机将每个队列的 DMA 描述符列表首地址写入网卡，并启动队列 DMA。网卡 DMA 被主机启动后，根据主机写入的描述符列表首地址读取每个队列连续两个描述符，仅在首次读取描述符时取两个，其后每次完成一个描述符并更新后再读取下一个。则可以实现网卡上的描述符缓存，保证任意时刻都有描述符待命，以防止读取描述符过程中引起丢包。

2) 从内存到硬盘。

使用 linux 内核的存储外设接口。用户进程通过系统调用 write()/read() 向磁盘写数

据，在内核中文件系统将底层硬盘外设完全抽象化。统一化的调用接口为分层设计：

- 虚拟文件系统层（Virtual Filesystem Layer, VFS）。系统调用 `read()` 会触发 VFS 函数，传递文件描述符和偏移量等信息。VFS 确定请求的数据是否已在内存缓冲区；若不存在，确定如何继续执行读操作。
- 映射层（Mapping Layer）。假设内核必须从块设备上读取数据，这样内核就必须确定数据在物理设备上的位置。这由映射层完成。
- 通用块设备层（Generic Block Layer）。内核通过 GBL 在块设备上执行读操作，启动 I/O 动作，完成数据传输。
- I/O 调度层（I/O Scheduler Layer）。在 GBL 之下是 I/O 调度层，根据内核的调度策略，对等待的 I/O 队列排序。
- 块设备驱动（Block Device Driver）。BDD 通过向磁盘控制器发送响应指令，执行真正的数据传输。

本系统实现了流量捕获与回放的吞吐性能为 1Gbps，其中捕获步骤中向硬盘写数据是系统最大的瓶颈，这是因为传统大容量机械式硬盘的写入速度慢通常只有 400Mbps 左右。为日后扩容性能，本文主要分析当前硬盘操作的相关技术发展现状。

在系统写操作（`sys_write()`）调用结束后，此时要写入的数据其实并没有被立即真正写入磁盘。已写入数据首先被拷贝到内核的缓冲区，并将相应的页标记为脏页。由于系统都需要高速缓存的支持，写操作实际上会被高速缓存延迟进行。当页高速缓存中的数据较后台存储的数据新的时候，该数据被称为脏数据。在内存中累计起来的脏页须最终被统一写回到磁盘，当发生以下两种情况时，写磁盘被触发：

- 当空闲内存低于一个特定的阈值时，内核必须将脏页写入磁盘，以释放内存空间。
- 当脏页在内存中主流时间超过特定阈值时，内核必须将超过的脏页写回磁盘，以确保脏页不会无限期驻留内存。

进行间隔同步工作的进程名叫 `pdflush`。`pdflush` 机制存在的问题：在多磁盘系统中，`pdflush` 管理所有磁盘的页/缓冲，从而导致一定程度的 I/O 瓶颈。从内核 2.6.32 开始，终止了 `pdflush` 机制，改成了 `bdi_writeback` 机制。新机制为每个磁盘都创建了一个专门线程，负责这个磁盘的页缓存数据刷新，从而实现了每个磁盘数据刷新程序在线程级别的分离，以提高 I/O 性能。另外目前固态硬盘（SSD）技术发展迅速，单块固态硬盘写入速度可高达 10Gbps，因此为日后系统升级铺平了道路。

4.6.2 硬件

为实现高精度的流量时间标注以及回放功能，本文实现了一款带有 PCIe Gen2 × 8 主机接口的基于 FPGA 的智能网卡 ONetCard（图4-17示）。

该网卡的核心器件使用 Xilinx 公司 FPGA 产品 XC7K325T，约有 33w 个可编程逻辑单元。为支持高速随机缓存和查找功能，FPGA 外部搭配了一块 9MB 的 QDR SRAM，以及 72MB 的 RLDRAM。网络接口部分有四个 G 比特电口和两路 SFP+ 万兆光口。PCIe



图 4-17 用于测试基于 FPGA 网卡的流量捕获功能的 ONetCard 实验平台

提供 Gen2×8 标准，总带宽达 40Gbps。板卡预留有 FMC 接口，可外扩高速串行总线的设备或接口，例如 TCAM 查找表、存储器、网络接口等。板卡尺寸与 NetFPGA 类似，但是能够提供的可编程逻辑数目更多，存储器种类更丰富。本文中的实验板卡可为各类网络计算加速应用提供工程实践支持。

4.7 系统评价

4.7.1 网络流量捕获与回放系统评估

基于 ISD14.1 及其自带仿真工具 isim，本文对数据通路中各个子模块分别进行功能仿真验证，最后通过上板测试符合时序约束的逻辑，系统运行在 125MHz 频率下。

1) 增加时间戳

该模块为进入数据通路的数据包添加时间戳信息，输入数据和输出数据的四组信号如图4-18所示，该模块在一级包头后面添加了控制字 FE 与对应的时间计数器数值。

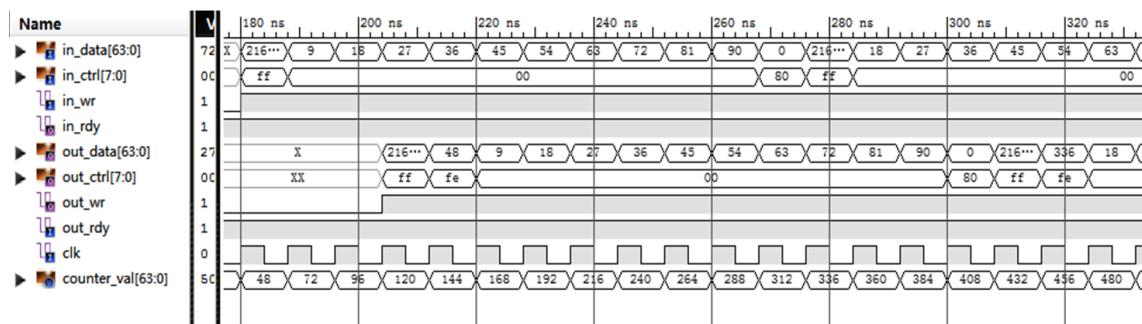


图 4-18 包头添加时间戳仿真结果

2) 回放数据包时间戳比较

验证时间戳排序比较器的功能，如图4-19最上方是四个时间戳 FIFO 的数值。分别是 800、1000、1200、4400 对应 1、2、3、4 路 DMA 写入网卡的队列。第一路时间戳为 800 是时间最小的，应该最早被输出。cmp_port_num 代表比较后的输出队列序号，可以看到有效的一个值是 0。这个数据将送给调度模块，控制其向后传送数据的通道来源，最终由 out_data 等信号输出端口输出按时间顺序排列的数据包。

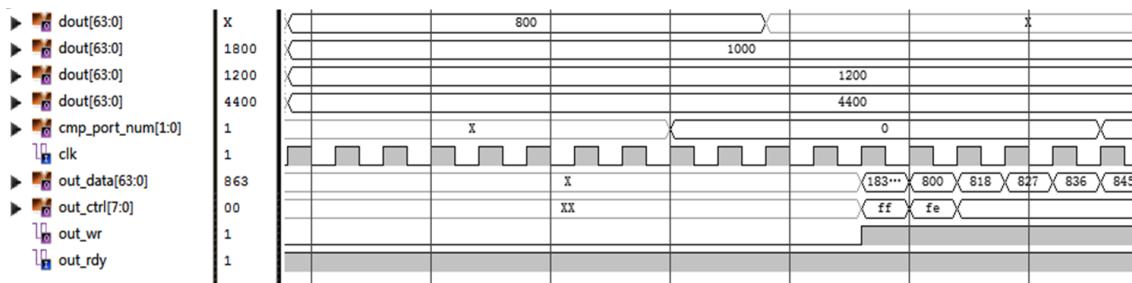


图 4-19 时间戳比较器测试仿真结果

在样机测试中，实验使用发包仪向板卡 1Gbps 接口中发送了 127,503,023 个 64 字节最小数据包。流量分流机制均匀地向四个队列分流了全体数据包，四个队列共计接受数据包个数分别为，31875756, 31875755, 31875756, 31875756。在回放测试中我们从四块硬盘中分别向网卡发送了 44160639, 44160639, 44340794, 44316028 个数据包，共计 176978100 个数据包。数据包均按照正确的顺序向外发送，并且无丢包现象。

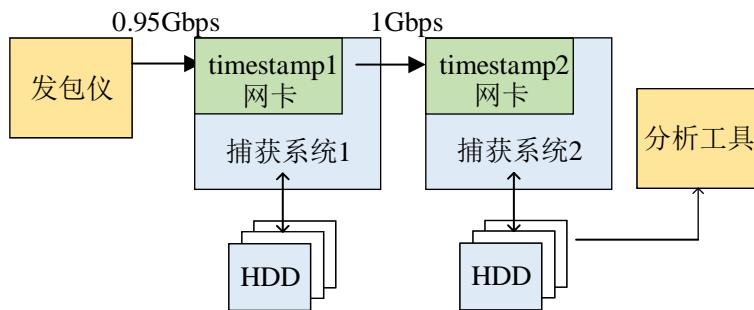


图 4-20 回放时间精度实验拓扑

3) 回放数据包精度

为测试数据包的回放真实精度，本实验设置了两台主机，如图4-20所示，每台主机内配置一个智能网卡。

- 主机 1，捕获数据包，并为数据包增加入队时间戳 (timestamp1)，以记录数据包真实到达时间分布。之后，对捕获的数据包进行回放。
- 主机 2，接受主机 1 发送的回放数据包，并第二次为包头增加时间戳 (timestamp2) 字段，并将所有的数据包捕获并存储。

因此本文可根据每个数据包头时间戳来得到其真实到达时间分布，以及经过本系统捕获和回放之后的输出时间分布。

如果系统不会引入时间误差则每对相邻数据包之间的时间间隔均相等。定义捕获数据包的时间间隔 $\Delta timestamp1_n = timestamp1_{n+1} - timestamp1_n$ ，回放数据包的时间间隔 $\Delta timestamp2_n = timestamp2_{n+1} - timestamp2_n$ ，时间系统误差 $time_bias_n = \Delta timestamp2_n - \Delta timestamp1_n$ 。

实验中本文分别测试了最大包（1518 字节）与最小包（64 字节），在 1Gbps 吞吐下的时间误差 $time_bias$ 。如图4-21所示，回放数据包中有将近 80% 可满足无时间误差。系统最大误差只有 32ns，即 4 个时钟周期时间长度。

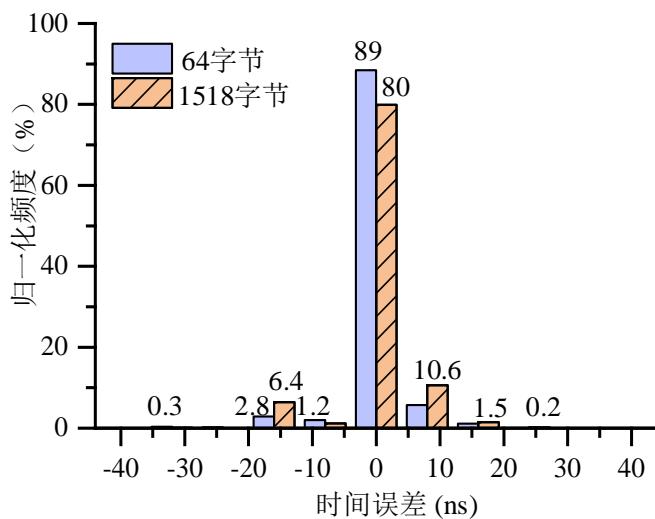


图 4-21 数据包回放时间精度概率分布

4.7.2 网络测量实时压缩系统评估

本实验^⑨目标为 100G 在线实时进行压缩测量，使用新一代的 Xilinx FPGA 智能网卡 Alveo U250。板卡有 2 个 100G 光纤网口，配有 PCIe Gen3×16，与主机互通接口带宽可达 128Gbps。

1) 测量性能

本文将测量基于 FPGA 智能网卡的压缩统计速度，并与 DISCO^[105] 算法，以及 CACTI^[106] 系统相对比。如图4-22本文对比了基于 x86 平台的 DISCO 算法，DISCO 运行时占用 3 个处理器核心的计算能力。另外本文还对比了相似工作 CACTI，CACTI 对 DISCO 在某种情形下做了优化。CACTI 利用少量（16 个）硬件高速计数器在数据平面内对流量做线性统计，之后定期将少量统计结果上报控制器，在控制器内对统计数量进行 DISCO 算法压缩。由于减小了压缩频率，因此 CACTI 能够获得比 DISCO 更高的吞吐性能。

对于 CACTI 系统，当流数目增多，硬件计数器需要频繁向 CPU 发送缓存更新请求。由于硬件计数器与 CPU 通信依然依赖于 PCIe 带宽（此类通信属突发传输），且系统不可占用过多通信管道资源，因此其系统设计时的最高通信容量只有 100kpps^[159]。一旦流数目超过了硬件计数器数量则系统会频繁请求更新，因此成为系统瓶颈理论性能大幅降低。本文实现的压缩统计最高需要耗费 3 个时钟周期，且对于同一条流的测量更新是阻塞的（更新时无法继续处理同一条流的后续请求），因此在测试单条流的测量时，性能只有理论最优的三分之一。当流数目多于三种时，每个更新请求可以按流水线方式进入处理通路，因而可以做的每包的单周期吞吐，64 字节小包处理性能可达到 133Gbps。

2) 主机 CPU 资源消耗

DISCO 与 CACTI 的压缩过程均由 CPU 完成，当压缩任务请求频次增加，CPU 计算

^⑨ 本实验为作者在赛灵思亚太研究院访问实习期间完成，西安交通大学与赛灵思亚太研究院为共同第一完成单位。

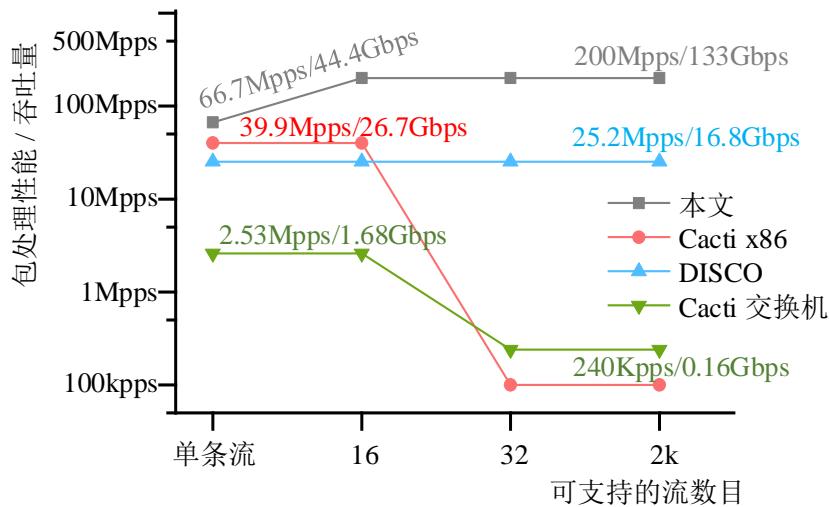


图 4-22 本文基于 FPGA 智能网卡压缩算法与其他压缩统计方法性能对比 (64 字节小包)

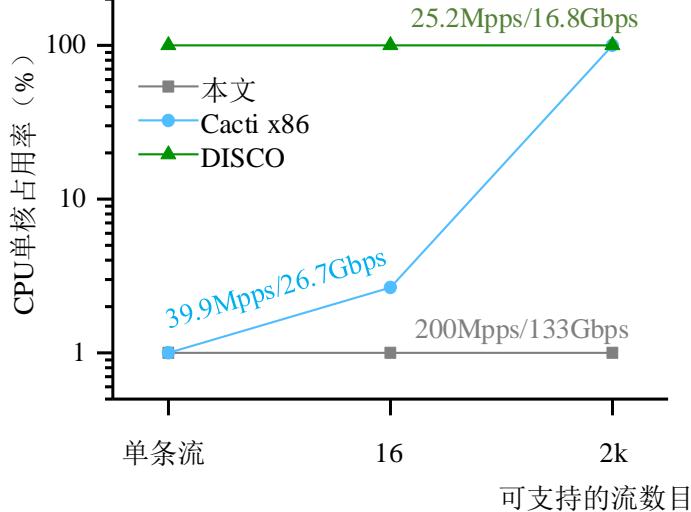


图 4-23 基于 FPGA 网卡硬件流量压缩统计系统可有效降低主机 CPU 消耗 (64 字节小包)

资源相应增加。如图4-23所示，实验为处理 64 字节小包时的最高性能情形下所对应的 CPU 占用率。本文将所有密集型计算任务均下放到基于 FPGA 的智能网卡，主机 CPU 只需定期查看计数器值即可，无需做额外复杂任务，从而将服务器 CPU 资源最大限度释放。

本文在软件平台内，使用 DISCO 算法完成了 100Gbps 流量压缩统计测试试验，总共占用了 CPU^⑩内 20 个核心。在智能网卡内运行本文优化后的压缩统计流水线 (133Gbps)。二者的功耗对比如表4-3所示，实验中着重对比了计算单元的功耗，并通过对照试验排除了服务器内主板、硬盘、电源模块等基础部件的功耗。

表 4-3 不同平台下压缩统计功能功耗对比 (单位：瓦特)

基于 CPU 的实现方案	FPGA 网卡
176.25	18.559

^⑩ Intel XEON E7-8894 v4

3) 存储资源消耗

通过不同方法实现 1 百万条流的统计所需要的存储空间如图4-24，对一条流进行简单线性叠加计数只需要开辟流对应的存储空间即可，在未优化的情形下一般需要为每一条流开辟 32bits 计数空间，然而此空间有固定上限（4GB），一旦超出需要开辟更大空间，或将记录值提前上报并清空。可压缩的计数方式上限较高，本文在压缩率达到 38% 时，计数上限可达 100GB，比固定上限值大 25 倍。可压缩计数方式可以灵活且低成本地扩展计数上限。DISCO、CACTI 等压缩算法不同设计方式会额外占用不同大小的固定空间（计算表），但总体差别不显著。

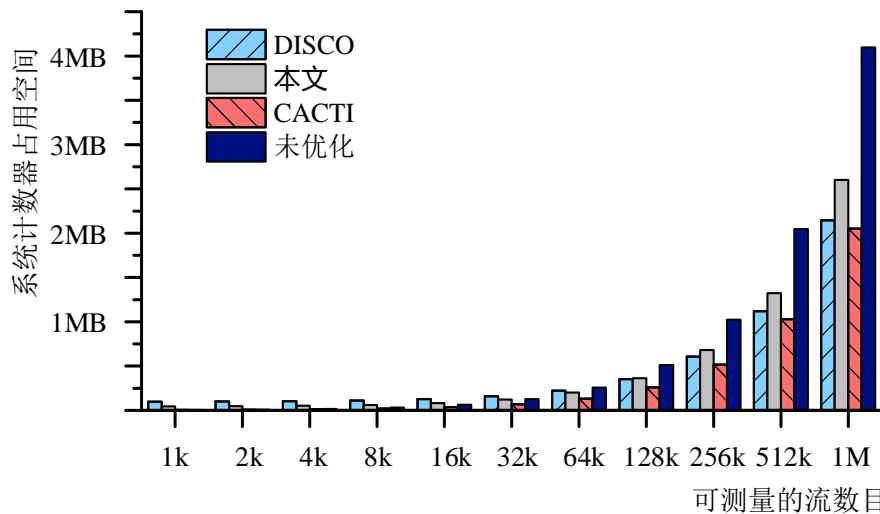


图 4-24 在不同流个数下的压缩计数存储空间占用

4) 测量误差

本文证明基于 FPGA 的硬件压缩是无偏估计，在统计学中，每一次估计都有估计偏差，本文利用前文^[105]提出的估计公式来表示估计偏差，且已经证明变异系数有如下关系：

$$COV(T(S)) = \sqrt{\frac{Var[T(S)]}{E^2[T(S)]}} = \sqrt{\frac{1 - 1/n}{2} \times (b - 1)} \quad (4-27)$$

其中 S 为随机变量，代表压缩后的统计值。 $T(S)$ 为由随机变量 S 估计的真实统计值。则变异系数即为标准差与平均值之比，变异系数常用于随机过程统计量偏差与误差分析。本文证明压缩估计方法为无偏估计，因而对于同一组序列的估计平均值为常数。则变异系数值越小说明估计值的误差错误比例越低。

如图4-25所示，在原始的 DISCO 算法中，由于小流的抽样频率低，抽样统计在处理小流量时的误差会相对较大。本文在数据流量较小阶段采用线性统计，误差率为零，极大地修复了基于抽象压缩算法的准确性。

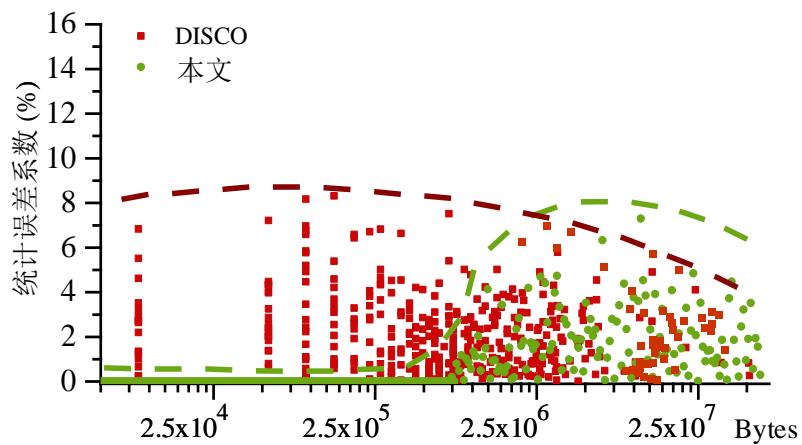


图 4-25 对不同大小 (Bytes) 的流量估计时的变异系数分布

4.8 本章小结

网络测量是众多网络功能的基础，针对目前其缺乏统一的编程抽象，本文提出一种基于测量的可编程处理器，利用在硬件内部署的“统计 + 触发”的可编程抽象模型，通过软件配置的方式，令系统硬件支持高性能的网络安全、访问控制、拥塞探测、流量控制等多种场景。本文提出了一套基于智能网卡的硬件流水线系统，包括数据包捕获功能、测量系统和发送引擎。在硬件统计模块中，利用基于硬件的存储压缩算法解决高速缓存资源不足的问题。系统将主机端 CPU 高耗费的网络应用卸载到可编程智能网卡内，大大缓解了传统网络系统性能低、精度低等问题，为进一步扩展服务器网络性能提供了一种思路和范例。

本章将有关捕获和发送功能的网络协议栈时间控制精度，由目前的几微秒提高到 20 纳秒。本章提出一种基于硬件的带有压缩存储空间容量的流量统计工具，根据本文提出的针对硬件设计的“数据—计算”抽象方法，本文将目前基于软件的 25Gbps 性能卸载到硬件中并提升到 133Gbps 吞吐；同时可压缩相应的高性能存储空间 40% 的占用量，并显著改善了基于 DISCO 压缩算法的统计精度。根据本文提出的系列算法，将测量系统的灵活性、资源消耗率与高性能合理折中，本文所耗费的 CPU 资源比之前工作下降了 99%。

5 结论与展望

5.1 工作总结与主要成果

网络通信是构建当今社会的重要基础设施，现代网络在向软件定义、数据平面可编程的方向发展。网络架构变迁的核心是有一套可编程的可以映射上层控制逻辑的硬件数据平面。本文主要探索编程硬件的网络系统，使其能够满足快速迭代的网络创新需求，在能够提供与现阶段匹配的处理性能的同时兼顾系统安全性。

本文从网络系统的三个维度进行分析：（1）主机侧网络，在服务器网卡层面，基于 CPU 的智能网卡性能难以满足目前虚拟化技术和网络监管细粒度化的发展需求。（2）交换侧网络，在核心网骨干网层面，基于 ASIC 的转发平面不足以提供网络处理的高灵活性。由于在成本、性能之间平衡困难，网络工程师的创新空间受到了限制。（3）控制面与数据面交互，硬件流表是一种高效且昂贵的网络转发核心部件，在软件定义网络时代流表稀缺性更加突出。由于流数目和流量的快速增长，控制平面针对流表的操作导致大量控制通信开销。易导致网络鲁棒性差，易形成安全隐患。现场可编程门阵列（FPGA）器件得到快速发展，以可编程硬件技术为首的异构架构已经大量融合到网络领域，带来高用户可定制能力的同时也能保证了一定的处理性能。

第一，SDN 网络硬件流表可扩展性研究。由可编程网卡和交换机组成的数据平面内，最重要的资源是流表资源。本文从 SDN 网络全局视野出发，着手解决流表资源匮乏的问题。本文分析不同的流量规模和特征，以及系统多模块之间的互联协议，提出一种转发设备节点之间的流表共享机制。本文基于流量分布规律以及理论分析，提出了流表共享机制 FTS，扩展了 OpenFlow 交换机的 Table-Miss 处理方式。可以有效地缓解流表溢出现象带来的巨大网络通信性能损失，实现了数据平面应对突发流量时的稳定性。

第二，本文为增强数据平面转发设备的网络随路计算能力，提出了一种自适应交换系统（Adaptive Switch Architecture, AS）。AS 通过保留 ASIC 交换芯片中的交换网络结构以确保高性能，并且将网络数据可编程性卸载到 FPGA 可编程硬件中。本文在 AS 架构中实现了三种难以在目前交换系统中实现的例子，集中体现了 AS 架构可为目前的交换芯片带来极高的灵活性。此外，本文设计了资源消耗可控的多级并行流水线，并使以上三个应用达到了数个 Tbps 的吞吐性能。在资源消耗可控的前提下，大规模提高可编程网络数据平面的包处理吞吐性能。

第三，本文研究可编程设备加速主机侧可编程网络的方法。论文将测量系统的可编程性抽象为基础的包个数统计和数据量缩统计方法。系统包括数据包捕获功能、测量系统和发送引擎。通过不同的软件调用，令系统硬件适用于高性能的网络安全、访问控制、流量控制、拥塞探测等多种场景。同时利用基于硬件的存储压缩算法，在节约 38% 的硬件存储空间情况下，相较于软件的处理方式系统吞吐率提升 8 倍、系统的处

理能耗节约 90%。在满足网络功能不改变的前提下，证明利用基于 FPGA 的智能网卡能有效地提升服务器的网络性能、降低抖动和提高效率。

综上全文，论文完成了对于可编程数据平面在网络各个层次中的性能优化、灵活性提升以及安全性保障。

5.2 研究内容展望

5.2.1 数据平面可编程网络

开放的网络数据平面概念已经催生出了新一轮的网络创新热潮，软件定义网络由数据平面控制平面分离转向全可定制化方向发展。学术界在可编程数据平面议题内探讨了大量的新应用新方法，为可编程数据平面展开了光明的发展前途。产业界在可编程数据平面的性能提升上持续耕耘。二者互相促进，为开放网络数据平面的标准化打下坚实的基础。

可编程数据平面中性能与灵活性是一对主要的矛盾，不同的需求场景下选取不同的实现方法，都体现了对这两方面的折中。本文发现异构的架构最终才有可能平衡各方面的需求。现场可编程门阵列是一种通用的数字逻辑技术，它可以实现完整的可编程能力。但受限于 FPGA 的架构，在网络处理中其性能并不占优。如何设计一种为网络专用的可编程硬件逻辑技术，来彻底解决性能与灵活的矛盾点，也许会成为一个崭新的方向。但这需要大量网络、芯片技术研究人员共同努力，也需要提升芯片制造和设计能力，是一个庞大的系统工程。

5.2.2 网络资源与计算能力

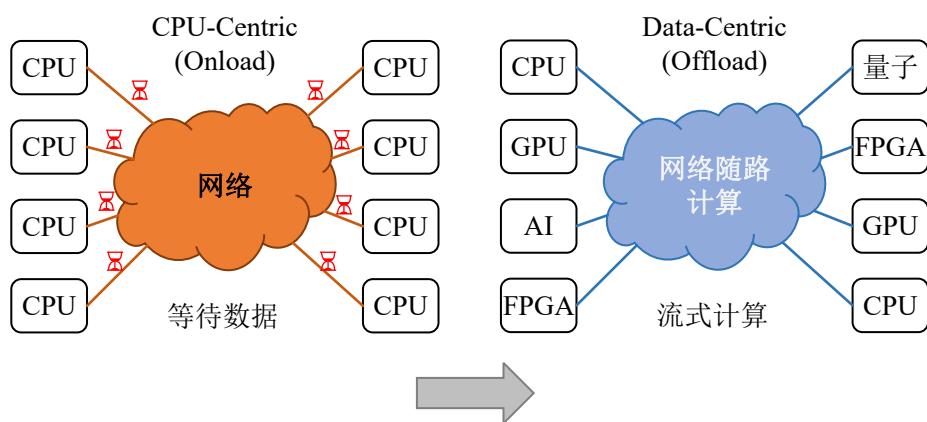


图 5-1 网络与计算的深度融合

今后，网络不只是承载通信功能，它还是一个对信息进行融合、智能计算和分发的整体设施。网络当中依然有方方面面的问题需要解决，例如转发时延、流量吞吐、网络随路计算等。不同问题的解决，都需要提出一种新的网络架构。对于转发时延和吞吐敏感的应用，人们利用 CDN（内容分发网络）和边缘计算来提升信息的快速可达性。对于计算能力来说，由于摩尔定律失效，人们不可能只依靠简单堆叠 CPU 来满足需求。异构计算（GPU、FPGA、ASIC）一方面是解决未来计算量增长的重要途径，另外也是

有效控制服务器数量规模的重要手段。让专业的设备处理专用任务这种概念也变得越来越清晰。网络在这种变化当中也可以起到粘合剂的效果，如图5-1所示，AI应用以及训练任务要求网络提供快速和高效的数据转发，传统以分布式CPU为核心的网络体系中等待数据传输的过程拉低了整体处理效率。网络随路计算、流式计算的概念都给网络应用加速提供了不同的可选车道。以各种不同计算技术（异构计算、量子信息计算）为核心，网络会智能化地指导数据流的最佳流动路径。网络的可编程化由最初的数据平面可编程变为了最终对计算库和数据包处理流程的可编程。

致 谢

写到这里意味着已经完成了研究生阶段全部工作，回顾读博士 7 年的时间，我于专业领域的知识与技术都得到了不小的进步。几千天在人生中不长，但也不短，一夜夜的点灯熬油、一行行的实验代码以及一篇篇文章专著都在我人生中留下了难忘的记忆。

首先感谢我一不留神就呆了 10 年的母校西安交通大学，她严谨的办学理念、优良的学风传统以及十六字校训为我打下为人为学的根基，也给了我深入地思考问题的空间。

导师是学海中的灯塔，感谢邹建华导师对我悉心照顾与学业上的帮助。感谢管晓宏院士为我的研究生涯指明方向，特别感谢您对我博士论文提出认真的修改意见，您对学术论文表达高屋建瓴令我启发深刻，您的每一句教导使我受益匪浅。感谢胡成臣教授对我科研和学习上的指导，他敏锐的思维和富有洞察力的视野给我深刻启迪。

在课题组的这段时间，交到了一辈子的朋友，一同学习也一直向大家学习：张帆、孙秀文、许琛、郑鹏、王瑞龙。

还要感谢在新加坡赛灵思亚太研究院的导师和同事们：Yan、Henry、Nguyen，近两年的海外访问实习经历令我的眼界及职业素养有长足进步。

感谢一直关心我的父母，你们的奉献、支持、鼓励和陪伴给了我莫大动力。

博士不只是一个头衔，更要有一种自主科研、分析和学习的能力。以开放的心态乐于接收新的知识碰撞思想的火花。善于发现自己的不足并富有探索精神，才是符合新时代青年的特征，愿为祖国繁荣昌盛而奋斗。

人生难得一位伴侣和知己，瑾以本文献给叶乃馨女士。

参考文献

- [1] 华为公司年报[EB/OL]. 2019. https://www-file.huawei.com/-/media/corporate/pdf/annual-report/annual_report_2019_cn.pdf.
- [2] 思科公司互联网发展跟踪白皮书（2018-2023）[EB/OL]. 2019. <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>.
- [3] GUBBI J, BUYYA R, MARUSIC S, et al. Internet of things (iot): A vision, architectural elements, and future directions[J]. Future generation computer systems, 2013, 29(7): 1645-1660.
- [4] HASHEM I A T, YAQOOB I, ANUAR N B, et al. The rise of “big data” on cloud computing: Review and open research issues[J]. Information systems, 2015, 47: 98-115.
- [5] LI Y, MIAO R, LIU H H, et al. Hpcc: High precision congestion control[M]//Proceedings of the ACM Special Interest Group on Data Communication. [S.l.: s.n.], 2019: 44-58.
- [6] SINGH A, ONG J, AGARWAL A, et al. Jupiter rising: A decade of clos topologies and centralized control in google’s datacenter network[J]. ACM SIGCOMM computer communication review, 2015, 45(4): 183-197.
- [7] ALIZADEH M, GREENBERG A, MALTZ D A, et al. Data center tcp (dctcp)[C]//Proceedings of the ACM SIGCOMM 2010 conference. [S.l.: s.n.], 2010: 63-74.
- [8] WU H, JU J, LU G, et al. Tuning ecn for data center networks[C]//Proceedings of the 8th international conference on Emerging networking experiments and technologies. [S.l.: s.n.], 2012: 25-36.
- [9] MCKEOWN N, ANDERSON T, BALAKRISHNAN H, et al. Openflow: enabling innovation in campus networks[J]. ACM SIGCOMM Computer Communication Review, 2008, 38(2): 69-74.
- [10] JAIN S, KUMAR A, MANDAL S, et al. B4: Experience with a globally-deployed software defined wan[J]. ACM SIGCOMM Computer Communication Review, 2013, 43(4): 3-14.
- [11] NUAGENETWORKS. Digitally transforming your network[EB/OL]. 2020. <https://www.nuagenetworks.net/>.
- [12] CENTEC. 盛科公司详情[EB/OL]. 2020. <http://www.centecnetworks.com/cn/About.asp?ID=24>.
- [13] 人民邮电报. 中国联通完成国内首个基于 SDN 与 SR 的互联网智能差异化服务商用部署[EB/OL]. 2019. <https://tech.sina.com/roll/2019-11-07/doc-iicezuev7740921.shtml>.
- [14] ZHANG P, LI H, HUC, et al. Mind the gap: Monitoring the control-data plane consistency in software defined networks[C]//Proceedings of the 12th International on Conference on emerging Networking EXperiments and Technologies. [S.l.: s.n.], 2016: 19-33.
- [15] ZHANG P, XU S, YANG Z, et al. Foces: Detecting forwarding anomalies in software defined networks[C]//2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). [S.l.]: IEEE, 2018: 830-840.
- [16] CASADO M, MCKEOWN N, SHENKER S. From ethane to sdn and beyond[J]. ACM SIGCOMM Computer Communication Review, 2019, 49(5): 92-95.
- [17] SHAHBAZ M, CHOI S, PFAFF B, et al. Pisces: A programmable, protocol-independent software switch[C]//Proceedings of the 2016 ACM SIGCOMM Conference. [S.l.: s.n.], 2016: 525-538.
- [18] BOSSHART P, DALY D, GIBB G, et al. P4: Programming protocol-independent packet processors [J]. ACM SIGCOMM Computer Communication Review, 2014, 44(3): 87-95.
- [19] BOSSHART P, GIBB G, KIM H S, et al. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn[J]. ACM SIGCOMM Computer Communication Review, 2013, 43(4): 99-110.

- [20] HONDA M, HUICI F, LETTIERI G, et al. mswitch: a highly-scalable, modular software switch[C]// Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research. [S.l.: s.n.], 2015: 1-13.
- [21] MIAO R, ZENG H, KIM C, et al. Silkroad: Making stateful layer-4 load balancing fast and cheap using switching asics[C]//Proceedings of the Conference of the ACM Special Interest Group on Data Communication. [S.l.: s.n.], 2017: 15-28.
- [22] LAPOLLI Â C, MARQUES J A, GASPARY L P. Offloading real-time ddos attack detection to programmable data planes[C]//2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM). [S.l.]: IEEE, 2019: 19-27.
- [23] YANG T, JIANG J, LIU P, et al. Elastic sketch: Adaptive and fast network-wide measurements[C]// Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. [S.l.: s.n.], 2018: 561-575.
- [24] KIM C, SIVARAMAN A, KATTA N, et al. In-band network telemetry via programmable dataplanes [C]//ACM SIGCOMM. [S.l.: s.n.], 2015.
- [25] JIN X, LI X, ZHANG H, et al. Netcache: Balancing key-value stores with fast in-network caching [C]//Proceedings of the 26th Symposium on Operating Systems Principles. [S.l.: s.n.], 2017: 121-136.
- [26] SCIENCE N C. Rmt and p4 notes[EB/OL]. 2018. https://cs.nyu.edu/~anirudh/CSCI-GA.2620-001/lectures/lec8_rmt_p4.txt.
- [27] CHOUE S, FINGERHUT A, MA S, et al. drmt: Disaggregated programmable switching[C]// Proceedings of the Conference of the ACM Special Interest Group on Data Communication. [S.l.: s.n.], 2017: 1-14.
- [28] 国际数据公司 (IDC) . 上半年中国公有云市场[EB/OL]. 2019. https://www.idc.com/url.do?url=/getdoc/pdf_download.do?containerId=prCHC45634819&position=15&transactionId=39032154&term=&page=5&perPage=100.
- [29] IREASEARCH. 中国公有云服务市场跟踪[EB/OL]. 2020. <http://news.iresearch.cn/yx/2020/02/315730.shtml>.
- [30] HONG C Y, MANDAL S, AL-FARES M, et al. B4 and after: managing hierarchy, partitioning, and asymmetry for availability and scale in google's software-defined wan[C]//Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication. [S.l.: s.n.], 2018: 74-87.
- [31] CASADO M, FREEDMAN M J, PETTIT J, et al. Ethane: Taking control of the enterprise[J]. ACM SIGCOMM computer communication review, 2007, 37(4): 1-12.
- [32] AL-FARES M, RADHAKRISHNAN S, RAGHAVAN B, et al. Hedera: dynamic flow scheduling for data center networks.[C]//Nsdi: volume 10. [S.l.: s.n.], 2010: 89-92.
- [33] HELLER B, SEETHARAMAN S, MAHADEVAN P, et al. Elastictree: Saving energy in data center networks.[C]//Nsdi: volume 10. [S.l.: s.n.], 2010: 249-264.
- [34] ONF/TS-022. Optical transport protocol extensions[EB/OL]. 2015. https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2014/10/Optical_Transport_Protocol_Extensions_V1.0.pdf.
- [35] ARYAKA. Managed sd-wan for digital transformation[EB/OL]. 2017. <https://www.aryaka.com/aryaka-sd-wan-solutions-for-manufacturing/>.
- [36] ONF/TS-029. Mpls-tp openflow protocol extensions for sptn[EB/OL]. 2017. <https://3vf60mmveq1g8vzn48q2o71a-wpengine.netdna-ssl.com/wp-content/uploads/2017/07/MPLS-TP-OpenFlow-Protocol-Extensions-for-SPTN-1-0.pdf>.

- [37] CURTIS A R, MOGUL J C, TOURRILHES J, et al. Devoflow: Scaling flow management for high-performance networks[C]//Proceedings of the ACM SIGCOMM 2011 conference. [S.l.: s.n.], 2011: 254-265.
- [38] ONF. Openflow switch specification version 1.1[EB/OL]. 2011. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.1.0.pdf>.
- [39] YU M, REXFORD J, FREEDMAN M J, et al. Scalable flow-based networking with difane[J]. ACM SIGCOMM Computer Communication Review, 2010, 40(4): 351-362.
- [40] YAZICI V, SUNAY M O, ERCAN A O. Controlling a software-defined network via distributed controllers[J]. arXiv preprint arXiv:1401.7651, 2014.
- [41] ONF. Openflow switch specification version 1.2[EB/OL]. 2011. <https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.2.pdf>.
- [42] H.XIE, T.TSOU, H.YIN, et al. Use cases for alto with software defined networks[J]. Working Draft, IETF Secretariat, Internet-Draft draft-xiealto-sdn-extension-use-cases-01.txt,, 2012.
- [43] LIU C, RAGHURAMU A, CHUAH C N, et al. Piggybacking network functions on sdn reactive routing: A feasibility study[C]//Proceedings of the Symposium on SDN Research. [S.l.: s.n.], 2017: 34-40.
- [44] AZAB M, FORTES J A. Towards proactive sdn-controller attack and failure resilience[C]//2017 International Conference on Computing, Networking and Communications (ICNC). [S.l.]: IEEE, 2017: 442-448.
- [45] TOONK A. Linux kernel and measuring network throughput.[EB/OL]. 2020. <https://medium.com/d-evops-dudes/linux-kernel-and-measuring-network-throughput-547c3b68c4d2>.
- [46] BERNAT V. Performance progression of ipv4 route lookup on linux[EB/OL]. 2017. <https://vincent.bernat.ch/en/blog/2017-performance-progression-ipv4-route-lookup-linux>.
- [47] KANNAN K, BANERJEE S. Compact tcam: Flow entry compaction in tcam for power aware sdn [C]//International conference on distributed computing and networking. [S.l.]: Springer, 2013: 439-444.
- [48] 周亚东, 陈凯悦, 冷俊园, 等. 软件定义网络流表溢出脆弱性分析及防御方法[J]. 西安交通大学学报, 2017(10): 53-58.
- [49] 郑鹏, 胡成臣, 李昊. 基于流量特征的 OpenFlow 南向接口开销优化技术[J]. 计算机研究与发展, 2018, 55(2): 346-357.
- [50] APISDORF J Z, SANDBOTE S B. System and method for instruction-level parallelism in a programmable multiple network processor environment[M]. [S.l.]: Google Patents, 2005.
- [51] ENNALS R, SHARP R, MYCROFT A. Task partitioning for multi-core network processors[C]// International Conference on Compiler Construction. [S.l.]: Springer, 2005: 76-90.
- [52] SOURLAS V, GKATZIKIS L, FLEGKAS P, et al. Distributed cache management in information-centric networks[J]. IEEE Transactions on Network and Service Management, 2013, 10(3): 286-299.
- [53] SCHUH K, HAWKINSON C, RUPLE S, et al. Distributed cache between servers of a network[M]. [S.l.]: Google Patents, 2007.
- [54] 中国工业信息化部. 2012 年使用 4M 宽带的用户将过半[EB/OL]. 2011. http://www.gov.cn/jrzq/2012-04/01/content_2104826.htm.
- [55] DE CARLI L, PAN Y, KUMAR A, et al. Plug: flexible lookup modules for rapid deployment of new protocols in high-speed routers[C]//Proceedings of the ACM SIGCOMM 2009 conference on Data communication. [S.l.: s.n.], 2009: 207-218.

- [56] ANWER M B, MOTIWALA M, TARIQ M B, et al. Switchblade: a platform for rapid deployment of network protocols on programmable hardware[C]//Proceedings of the ACM SIGCOMM 2010 conference. [S.l.: s.n.], 2010: 183-194.
- [57] INTEL. Intel ethernet switch fm6000 series[EB/OL]. 2013. <https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/ethernet-switch-fm6000-series-brief.pdf>.
- [58] BAREFOOT. The world's fastest p4 programmable ethernet switch asics[EB/OL]. 2017. <https://barefootnetworks.com/products/brief-tofino/>.
- [59] BAREFOOT. Second generation of world's fastest p4 programmable ethernet switch asics[EB/OL]. 2020. <https://www.barefootnetworks.com/products/brief-tofino-2/>.
- [60] NAOUS J, ERICKSON D, COVINGTON G A, et al. Implementing an openflow switch on the netfpga platform[C]//Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems. [S.l.: s.n.], 2008: 1-9.
- [61] YABE T. Openflow implementation on netfpga-10g design document[M]. [S.l.]: Stanford University, 2011.
- [62] ZILBERMAN N, AUDZEVICH Y, COVINGTON G A, et al. Netfpga sume: Toward 100 gbps as research commodity[J]. IEEE micro, 2014, 34(5): 32-41.
- [63] HAN J H, MUNDKUR P, ROTSOS C, et al. Blueswitch: Enabling provably consistent configuration of network switches[C]//2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS). [S.l.]: IEEE, 2015: 17-27.
- [64] LI B, TAN K, LUO L, et al. Clicknp: Highly flexible and high performance network processing with reconfigurable hardware[C]//Proceedings of the 2016 ACM SIGCOMM Conference. [S.l.: s.n.], 2016: 1-14.
- [65] WANG H, SOULÉ R, DANG H T, et al. P4fpga: A rapid prototyping framework for p4[C]//Proceedings of the Symposium on SDN Research. [S.l.: s.n.], 2017: 122-135.
- [66] XILINX. Packet processor smartcore[EB/OL]. 2017. <https://www.xilinx.com/support/documentation/navigation/development-tools/software-development/sdnet.html>.
- [67] FIRESTONE D, PUTNAM A, MUNDKUR S, et al. Azure accelerated networking: Smartnics in the public cloud[C]//15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18). [S.l.: s.n.], 2018: 51-66.
- [68] INTEL. Ixp4xx product line of network processors[EB/OL]. 2010. <https://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/ixp4xx-product-line-network-processors-spec-update.pdf>.
- [69] OPENSWITCH. Cavium-xpliant® family of programmable ethernet switches[EB/OL]. 2010. <https://www.openswitch.net/cavium/>.
- [70] NETRONOME. Agilio cx 2x40gbe intelligent server adapter[EB/OL]. 2016. <http://colfaxdirect.com/store/pc/catalog/Agilio-CX-2x40GbE.pdf>.
- [71] PAGIANTZIS K, SHEIKHOLESLAMI A. Content-addressable memory (cam) circuits and architectures: A tutorial and survey[J]. IEEE journal of solid-state circuits, 2006, 41(3): 712-727.
- [72] FELDMAN A, MUTHUKRISHNAN S. Tradeoffs for packet classification[C]//Proceedings IEEE INFOCOM 2000. Conference on computer communications. Nineteenth annual joint conference of the IEEE computer and communications societies (Cat. No. 00CH37064): volume 3. [S.l.]: IEEE, 2000: 1193-1202.
- [73] KOGAN K, NIKOLENKO S, ROTTENSTREICH O, et al. Sax-pac (scalable and expressive packet classification)[C]//Proceedings of the 2014 ACM conference on SIGCOMM. [S.l.: s.n.], 2014: 15-26.

- [74] SRINIVASAN V, SURI S, VARGHESE G. Packet classification using tuple space search[C]// Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication. [S.l.: s.n.], 1999: 135-146.
- [75] GREENHALGH A, HUICI F, HOERDT M, et al. Flow processing and the rise of commodity network hardware[J]. ACM SIGCOMM Computer Communication Review, 2009, 39(2): 20-26.
- [76] FLOWFORWARDING. Link is not closed[EB/OL]. 2013. <https://flowforwarding.github.io/LINC-Switch/>.
- [77] FLOODLIGHT. Open source project to support openflow on a range of physical and now virtual switch platforms[EB/OL]. 2013. <https://github.com/floodlight/indigo>.
- [78] IN BRAZIL E I C. Basic openflow software switch[EB/OL]. 2013. <https://cpqd.github.io/ofsoftswitch13/>.
- [79] SNABB. Snabb switch:a simple and fast packet networking toolkit[EB/OL]. 2015. <https://github.com/snabbco/snabb>.
- [80] PFAFF B, PETTIT J, KOPONEN T, et al. The design and implementation of open vswitch[C]//12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15). [S.l.: s.n.], 2015: 117-130.
- [81] MOLNÁR L, PONGRÁCZ G, ENYEDI G, et al. Dataplane specialization for high-performance openflow software switching[C]//Proceedings of the 2016 ACM SIGCOMM Conference. [S.l.: s.n.], 2016: 539-552.
- [82] PANDA A T, HAN S, JANG K, et al. Netbricks: Taking the v out of {NFV}[C]//12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16). [S.l.: s.n.], 2016: 203-216.
- [83] CONSORTIUM P L, et al. Behavioral model (bmv2)[J]. URL: <https://github.com/p4lang/behavioral-model> [cited 2020-01-21], 2018.
- [84] AZURE M. Open source network operating system[EB/OL]. 2016. <https://azure.github.io/SONiC/>.
- [85] DALTON M, SCHULTZ D, ADRIAENS J, et al. Andromeda: Performance, isolation, and velocity at scale in cloud network virtualization[C]//15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18). [S.l.: s.n.], 2018: 373-387.
- [86] HAN S, JANG K, PARK K, et al. Packetshader: a gpu-accelerated software router[J]. ACM SIGCOMM Computer Communication Review, 2010, 40(4): 195-206.
- [87] KALIA A, ZHOU D, KAMINSKY M, et al. Raising the bar for using gpus in software packet processing[C]//12th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 15). [S.l.: s.n.], 2015: 409-423.
- [88] GO Y, JAMSHERD M A, MOON Y, et al. Apunet: Revitalizing {GPU} as packet processing accelerator[C]//14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17). [S.l.: s.n.], 2017: 83-96.
- [89] GIBB G, VARGHESE G, HOROWITZ M, et al. Design principles for packet parsers[C]// Architectures for Networking and Communications Systems. [S.l.]: IEEE, 2013: 13-24.
- [90] SHINDE P, KAUFMANN A, ROSCOE T, et al. We need to talk about nics[C]//Presented as part of the 14th Workshop on Hot Topics in Operating Systems. [S.l.: s.n.], 2013.
- [91] COSTA P, DONNELLY A, ROWSTRON A, et al. Camoop: Exploiting in-network aggregation for big data applications[C]//Presented as part of the 9th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 12). [S.l.: s.n.], 2012: 29-42.

- [92] SAPIO A, ABDELAZIZ I, ALDILAIJAN A, et al. In-network computation is a dumb idea whose time has come[C]//Proceedings of the 16th ACM Workshop on Hot Topics in Networks. [S.l.: s.n.], 2017: 150-156.
- [93] MAI L, RUPPRECHT L, ALIM A, et al. Netagg: Using middleboxes for application-specific on-path aggregation in data centres[C]//Proceedings of the 10th ACM International Conference on Emerging Networking Experiments and Technologies. [S.l.: s.n.], 2014: 249-262.
- [94] GRAHAM R L, BUREDDY D, LUI P, et al. Scalable hierarchical aggregation protocol (sharp): a hardware architecture for efficient data reduction[C]//2016 First International Workshop on Communication Optimizations in HPC (COMHPC). [S.l.]: IEEE, 2016: 1-10.
- [95] LIU M, LUO L, NELSON J, et al. Incbricks: Toward in-network computation with an in-network cache[C]//Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems. [S.l.: s.n.], 2017: 795-809.
- [96] SANVITO D, SIRACUSANO G, BIFULCO R. Can the network be the ai accelerator?[C]//Proceedings of the 2018 Morning Workshop on In-Network Computing. [S.l.: s.n.], 2018: 20-25.
- [97] SIRACUSANO G, BIFULCO R. In-network neural networks[EB/OL]. 2018. <https://arxiv.org/pdf/1801.05731.pdf>.
- [98] CROWLEY P, FRANKLIN M A, HADIMIOGLU H, et al. Network processor design: issues and practices: volume 1[M]. [S.l.]: Morgan Kaufmann, 2003.
- [99] ALLEN J R, BASS B M, BASSO C, et al. Ibm powernp network processor: Hardware, software, and applications[J]. IBM Journal of research and development, 2003, 47(2.3): 177-193.
- [100] PUTNAM A. Fpgas in the datacenter: Combining the worlds of hardware and software development [C]//Proceedings of the on Great Lakes Symposium on VLSI 2017. [S.l.: s.n.], 2017: 5-5.
- [101] D' ALCONZO A, DRAGO I, MORICHETTA A, et al. A survey on big data for network traffic monitoring and analysis[J]. IEEE Transactions on Network and Service Management, 2019, 16(3): 800-813.
- [102] ZHANG X, WU Z, ZHANG J, et al. An adaptive network traffic prediction approach for ldos attacks detection[J]. International Journal of Communication Systems, 2018, 31(5): e3505.
- [103] ESTAN C, KEYS K, MOORE D, et al. Building a better netflow[J]. ACM SIGCOMM Computer Communication Review, 2004, 34(4): 245-256.
- [104] TAHAEI H, SALLEH R, KHAN S, et al. A multi-objective software defined network traffic measurement[J]. Measurement, 2017, 95: 317-327.
- [105] HU C, LIU B, ZHAO H, et al. Discount counting for fast flow statistics on flow size and flow volume [J]. IEEE/ACM Transactions on Networking, 2013, 22(3): 970-981.
- [106] YANG J, HU C, ZHENG P, et al. Rethinking the design of openflow switch counters[C]//Proceedings of the 2016 ACM SIGCOMM Conference. [S.l.: s.n.], 2016: 589-590.
- [107] EINZIGER G, FRIEDMAN R, MANES B. Tinylfu: A highly efficient cache admission policy[J]. ACM Transactions on Storage (ToS), 2017, 13(4): 1-31.
- [108] TONE K, TSUTSUI M. Network dea: A slacks-based measure approach[J]. European journal of operational research, 2009, 197(1): 243-252.
- [109] QIAO S, HU C, GUAN X, et al. Taming the flow table overflow in openflow switch[C]//Proceedings of the 2016 ACM SIGCOMM Conference. [S.l.: s.n.], 2016: 591-592.
- [110] 乔思祎, 胡成臣, 李昊, 等. OpenFlow 交换机流表溢出问题的缓解机制[J]. 计算机学报, 2018, 41(9): 2003-2015.

- [111] GE X, LIU Y, DU D H, et al. Openanfv: Accelerating network function virtualization with a consolidated framework in openstack[J]. ACM SIGCOMM Computer Communication Review, 2014, 44(4): 353-354.
- [112] LI X, WANG X, LIU F, et al. Dhl: Enabling flexible software network functions with fpga acceleration [C]//2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). [S.l.]: IEEE, 2018: 1-11.
- [113] QIAO S, XU C, XIE L, et al. Network recorder and player: Fpga-based network traffic capture and replay[C]//2014 International Conference on Field-Programmable Technology (FPT). [S.l.]: IEEE, 2014: 342-345.
- [114] LIU M, PETER S, KRISHNAMURTHY A, et al. E3: energy-efficient microservices on smartnic-accelerated servers[C]//2019 {USENIX} Annual Technical Conference ({USENIX} {ATC} 19). [S.l.: s.n.], 2019: 363-378.
- [115] LE Y, CHANG H, MUKHERJEE S, et al. Uno: uniflying host and smart nic offload for flexible packet processing[C]//Proceedings of the 2017 Symposium on Cloud Computing. [S.l.: s.n.], 2017: 506-519.
- [116] KATTA N, ALIPOURFARD O, REXFORD J, et al. Infinite cacheflow in software-defined networks [C]//Proceedings of the third workshop on Hot topics in software defined networking. [S.l.: s.n.], 2014: 175-180.
- [117] KUŽNIAR M, PEREŠÍNI P, KOSTIĆ D. What you need to know about sdn flow tables[C]// International Conference on Passive and Active Network Measurement. [S.l.]: Springer, 2015: 347-359.
- [118] XILINX. Alveo u250 data center accelerator card[EB/OL]. 2020. <https://www.xilinx.com/products/boards-and-kits/alveo/u250.html>.
- [119] NETFPGA. A line-rate, flexible, and open platform for research, and classroom experimentation. [EB/OL]. 2020. <https://netfpga.org/site/#/about/>.
- [120] 开放网络基金会. OpenFlow1.5 协议规范[EB/OL]. 2014. <https://www.opennetworking.org/wp-content/uploads/2014/10/openflow-switch-v1.5.1.pdf>.
- [121] 乔思祎, 胡成臣, 李昊, 等. OpenFlow 交换机流表溢出问题的缓解机制[J]. 计算机学报, 2018, 41(9): 2003-2015.
- [122] MEINERS C R, LIU A X, TORNG E. Bit weaving: A non-prefix approach to compressing packet classifiers in tcams[J]. IEEE/ACM Transactions on Networking, 2011, 20(2): 488-500.
- [123] KANNAN K, BANERJEE S. Flowmaster: Early eviction of dead flow on sdn switches[C]// International Conference on Distributed Computing and Networking. [S.l.]: Springer, 2014: 484-498.
- [124] LI L E, MAO Z M, REXFORD J. Toward software-defined cellular networks[C]//2012 European workshop on software defined networking. [S.l.]: IEEE, 2012: 7-12.
- [125] BENSON T, ANAND A, AKELLA A, et al. Microte: Fine grained traffic engineering for data centers [C]//Proceedings of the Seventh COncference on emerging Networking EXperiments and Technologies. [S.l.: s.n.], 2011: 1-12.
- [126] CERRATO I, ANNARUMMA M, RISSO F. Supporting fine-grained network functions through intel dpdk[C]//2014 Third European Workshop on Software Defined Networks. [S.l.]: IEEE, 2014: 1-6.
- [127] KOPONEN T, CASADO M, GUDE N, et al. Onix: A distributed control platform for large-scale production networks.[C]//OSDI: volume 10. [S.l.: s.n.], 2010: 1-6.

- [128] KLEINROCK L. Queueing systems, volume 1: Theory: volume 66[M]. [S.l.]: wiley New York, 1975.
- [129] BOLCH G, GREINER S, DE MEER H, et al. Single station queueing systems: volume 6[M]. [S.l.]: wiley New York, 1998.
- [130] ALLESINA S, BONDAVALLI C. Wand: an ecological network analysis user-friendly tool[J]. Environmental Modelling & Software, 2004, 19(4): 337-340.
- [131] HUAWEI. 1800V 虚拟交换机[EB/OL]. 2018. <https://carrier.huawei.com/~/media/CN BG/Downloads/Product/Fixed%20Network/b2b/0920/1800-en.pdf>.
- [132] SANDVINE. Hyperscale data plane for next generation telco networks[EB/OL]. 2020. https://www.sandvine.com/hubfs/Sandvine_Redesign_2019/Downloads/2020/Datasheets/Network%20Optimization/Sandvine_DS_ActiveLogic.pdf.
- [133] CENTEC. Hybrid v580 sdn switch[EB/OL]. 2019. <http://www.centecnetworks.com/cn/DownView.asp?ID=2272&SortID=153>.
- [134] 高山渊, 蔡德忠, 赵晓雪, 等. 企业数字化基石-阿里巴巴云计算基础设施实践[M]. 北京市海淀区: 电子工业出版社, 2020.
- [135] HANDLEY M, RAICIU C, AGACHE A, et al. Re-architecting datacenter networks and stacks for low latency and high performance[C]//Proceedings of the Conference of the ACM Special Interest Group on Data Communication. [S.l.: s.n.], 2017: 29-42.
- [136] PONTARELLI S, BIFULCO R, BONOLA M, et al. Flowblaze: Stateful packet processing in hardware[C]//16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19). [S.l.: s.n.], 2019: 531-548.
- [137] ZERKANE S, ESPES D, LE PARC P, et al. Software defined networking reactive stateful firewall [C]//IFIP International Conference on ICT Systems Security and Privacy Protection. [S.l.]: Springer, 2016: 119-132.
- [138] XILINX. Understanding performance of pci express systems[EB/OL]. 2014. https://www.xilinx.com/support/documentation/white_papers/wp350.pdf.
- [139] ZHENG K, HU C, LU H, et al. A tcam-based distributed parallel ip lookup scheme and performance analysis[J]. IEEE/ACM Transactions on networking, 2006, 14(4): 863-875.
- [140] CORMEN T H, LEISERSON C E, RIVEST R L, et al. Introduction to algorithms[M]. [S.l.]: MIT press, 2009.
- [141] SUN C, BI J, CHEN H, et al. Sdpa: Toward a stateful data plane in software-defined networking[J]. IEEE/ACM Transactions on Networking, 2017, 25(6): 3294-3308.
- [142] KOHLER T, MAYER R, DÜRR F, et al. P4cep: Towards in-network complex event processing[C]// Proceedings of the 2018 Morning Workshop on In-Network Computing. [S.l.: s.n.], 2018: 33-38.
- [143] LU G, GUO C, LI Y, et al. Serverswitch: a programmable and high performance platform for data center networks.[C]//Nsdi: volume 11. [S.l.: s.n.], 2011: 2-2.
- [144] FIRESTONE D, PUTNAM A, MUNDKUR S, et al. Azure accelerated networking: Smartnics in the public cloud[C]//15th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 18). [S.l.: s.n.], 2018: 51-66.
- [145] YAN Y, SARIDIS G M, SHU Y, et al. All-optical programmable disaggregated data centre network realized by fpga-based switch and interface card[J]. Journal of Lightwave Technology, 2016, 34(8): 1925-1932.
- [146] XILINX CO. L. [EB/OL]. 2020. <https://www.xilinx.com/about/company-overview.html>.

- [147] XILINX. Smartnics for diverse workloads[EB/OL]. 2020. <https://www.xilinx.com/applications/data-center/network-acceleration.html#smartnics>.
- [148] INTEL. Fpga programmable acceleration card n3000 for networking[EB/OL]. 2020. https://plan.seek.intel.com/psg_WW_psgcom3_LPCS_EN_2019_PACN3000ProductBrief.
- [149] INTEL. 5G 前传边缘网络 FPGA (IP) 方案[EB/OL]. 2020. https://plan.seek.intel.com/5GFrontHaulGatedFormCN_LP?erpm_id=8235613&erpm_id=8235613&elq_cid=6511651.
- [150] XILINX. Stand alone nvme-of acceleration solution[EB/OL]. 2020. https://www.xilinx.com/publications/solution-briefs/partner/nvme-of_solutionbrief.pdf.
- [151] INTEL. SDN/NFV 低延迟 GRE 处理加速器[EB/OL]. 2020. https://www.intel.cn/content/dam/altera-www/global/zh_CN/pdfs/literature/wp/low-latency-gre-processing-accelerator-evaluation-cn.pdf.
- [152] INTEL. 电信解决方案 FPGA PAC N3000 助力在云环境中实现大容量 DDoS 防护[EB/OL]. 2020. <https://www.intel.cn/content/dam/www/programmable/cn/zh/pdfs/literature/solution-sheets/sb-high-capacity-ddos-protection-in-cloud-environments-cn.pdf>.
- [153] INTEL. 英特尔 FPGA 可编程加速卡 N3000 的 IPsec 加速解决方案[EB/OL]. 2020. https://www.intel.cn/content/dam/altera-www/global/zh_CN/pdfs/literature/solution-sheets/sb-accelerating-ipsec-arrive-technology-intel-fgpa-pac3000-cn.pdf.
- [154] XILINX. Vivado high-level synthesis accelerates ip creation by enabling c/c++ and system c specifications[EB/OL]. 2020. <https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html>.
- [155] XILINX. Vivado hls documentation[EB/OL]. 2020. <https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html#documentation>.
- [156] XILINX. Alveo u200 and u250 data center accelerator cards data sheet[EB/OL]. 2020. https://www.xilinx.com/support/documentation/data_sheets/ds962-u200-u250.pdf.
- [157] XILINX. Multiplier v12.0[EB/OL]. 2015. https://www.xilinx.com/support/documentation/ip_documentation/mult_gen/v12_0/pg108-mult-gen.pdf.
- [158] FERNANDES E L, ROJAS E, ALVAREZ-HORCAJO J, et al. The road to bofuss: The basic openflow userspace software switch[J]. Journal of Network and Computer Applications, 2020: 102685.
- [159] 杨骥. 细粒度可控可测可管的软件定义网络数据平面研究[D]. 中国, 陕西, 西安: 西安交通大学, 2017.

攻读学位期间取得的研究成果

论文：

- [1] **Qiao Siyi**, Hu Chengchen, Gordon Brebner, Zou Jianhua, Guan Xiaohong. Adaptive Switch: A Heterogeneous Switch Architecture for Network-Centric Computing[J].IEEE Communications Magazine.2020. (最具 SCI, IF=12.04, 正在出三审结果).
- [2] 乔思袆, 胡成臣, 李昊, 管晓宏, 邹建华.OpenFlow 交换机流表溢出问题的缓解机制 [J]. 计算机学报, 2018,41(9):2003-2015.(CCF A, EI: 20184406006908).
- [3] **Qiao Siyi**, Hu Chengchen, Guan Xiaohong, Zou Jianhua. Taming the flow table overflow in openflow switch[C].In Proceedings of the 2016 ACM SIGCOMM Conference. 2016: 591-592. (CCF A, EI: 20163702808112).
- [4] **Qiao Siyi**, Xu Chen, Xie Lei, Yang Ji, Hu Chengchen, Guan Xiaohong, Zou Jianhua. Network recorder and player: FPGA-based network traffic capture and replay[C].2014 International Conference on Field-Programmable Technology (FPT). IEEE, 2014: 342-345.(CCF C, EI: 20151900816423).

科研获奖：

- [1] 乔思袆（首位）.OpenFlow 交换机流表溢出问题的缓解机制. 第五届中国互联网学术年会最佳学生论文奖. 中国计算机学会互联网专业委员会.2016-08-15.

专利：

- [1] 胡成臣, 孙秀文, 乔思袆, 李昊. 一种基于 FPGA 平台的压缩流量模式匹配引擎及模式匹配方法 [P]. 陕西省: CN110865970A, 2020-03-06 (已受理) .
- [2] Hu Chengchen, Yang Ji, Zhang Yan, **Qiao Siyi**. A hardware architecture for passive network measurement based on flow statistics. US 专利. 2019.
- [3] 胡成臣, 杨骥, 龚志敏, 杨卫, 赵泓博, 乔思袆, 张丽山, 徐友庆, 吕伟男. 一种全可编程 SDN 高速网卡 [P]. 江苏: CN204392269U, 2015-06-10.
- [4] 胡成臣, 赵泓博, 吕伟男, 杨骥, 团哲恒, 史明, 乔思袆, 杨卫. 一种全可编程 SDN 交换机 [P]. 江苏: CN204168323U, 2015-02-18.

用于盲审的论文，只列出已发表学术论文的题目和刊物名称，可以备注自己为第几作者，及期刊影响因子。

声明

学位论文独创性声明 (1)

本人声明：所呈交的学位论文系在导师指导下本人独立完成的研究成果。文中依法引用他人的成果，均已做出明确标注或得到许可。论文内容未包含法律意义上已属于他人的任何形式的研究成果，也不包含本人已用于其他学位申请的论文或成果。

本人如违反上述声明，愿意承担以下责任和后果：

1. 交回学校授予的学位证书；
2. 学校可在相关媒体上对作者本人的行为进行通报；
3. 本人按照学校规定的方式，对因不当取得学位给学校造成的名誉损害，进行公开道歉；
4. 本人负责因论文成果不实产生的法律纠纷。

论文作者(签名): _____ 日期: _____ 年 _____ 月 _____ 日

学位论文独创性声明 (2)

本人声明：研究生_____ 所提交的本篇学位论文已经本人审阅，确系在本人指导下由该生独立完成的研究成果。

本人如违反上述声明，愿意承担以下责任和后果：

1. 学校可在相关媒体上对本人的失察行为进行通报；
2. 本人按照学校规定的方式，对因失察给学校造成的名誉损害，进行公开道歉；
3. 本人接受学校按照有关规定做出的任何处理。

指导教师(签名): _____ 日期: _____ 年 _____ 月 _____ 日

学位论文知识产权权属声明

我们声明，我们提交的学位论文及相关的职务作品，知识产权归属学校。学校享有以任何方式发表、复制、公开阅览、借阅以及申请专利等权利。学位论文作者离校后，或学位论文导师因故离校后，发表或使用学位论文或与该论文直接相关的学术论文或成果时，署名单位仍然为西安交通大学。

论文作者(签名): _____ 日期: _____ 年 _____ 月 _____ 日

指导教师(签名): _____ 日期: _____ 年 _____ 月 _____ 日

(本声明的版权归西安交通大学所有，未经许可，任何单位及任何个人不得擅自使用)